Fall 2019

Insider's Misuse Detection: From Hidden Markov Model to Deep Learning

Ahmed Saaudi

INSIDER'S MISUSE DETECTION: FROM HIDDEN MARKOV MODEL TO DEEP LEARNING

by

Ahmed Saaudi

Bachelor of Science
University of Baghdad 2007

Master of Science
University of Al-Nahrain 2011

Master of Science
University of South Carolina 2018

Master of Science
University of South Carolina 2019

Submitted in Partial Fulfillment of the Requirements

for the Degree of Doctor of Philosophy in

Computer Science and Engineering

College of Engineering and Computing

University of South Carolina

2019

Accepted by:

Csilla Farkas, Major Professor

Yan Tong, Major Professor

Jose M. Vidal, Committee Member

John R. Rose, Committee Member

Dezhi Wu, Committee Member

Cheryl L. Addy, Vice Provost and Dean of the Graduate School

ACKNOWLEDGMENTS

I am forever grateful to my advisors Professor Farkas and Professor Tong, for giving me the opportunity to work on this subject and finding new ideas to support my project. I am grateful for their guidance, support, knowledge, great experience, and advice, that led me to succeed.

I am especially grateful to my committee, Professor Dezhi Wu, Professor Jose M. Vidal, and Professor John R. Rose. Your input and advice helped me to become a better scholar.

To Mom - thanks for helping me achieve so much in my life. Your love, support, encouragement, and prayers have helped me get where I am today. Thanks to my sister Adwaa and my twin brother Hassanein for supporting me and taking care of our family in Iraq. Thanks to my father, may God give mercy on his soul, for supporting me during the hard times in Iraq.

A special thanks to my family, my wife Rifqa Al-ward, and my mother-in-law, Dr. Najwa Mohammed: your love and support have always lighted my way. Thanks to my son Ali, my daughter Hawraa, and my baby boy Yousuf, your hugs give me peace and encouragement. Also, a special thanks to my sister-in-law Qabas Al-ward for her support, encouragement, and help in proof reading some of my work.

I would like to extend my sincere thanks to my Iraqi sponsor Mohser for giving me this opportunity to complete my Ph.D. study in the United States of America.

A special thanks to my friends Saad K. Oudah and Hayder Abbood for their friendship and encouragement. Thanks to all Iraqi students at the University of South Carolina, our gatherings always gives me support and encouragement. A special thanks to my lab mates and colleagues, Husain Al-mulla, Zaid Alibadi, Emad Alsuwat, Hatim Alsuwat, Xianshan

Qu, and Tieming Geng for their help and support.

A special thanks to Research Computing staff, especially Bob Doran and Ben Torkian, for their help while running my experiments on the Hyperion Cluster.

A special thanks to all my friends and colleagues at Richland County, specially Ashley Powell, Ashiya Myers, Tammy Addy, Dr. John Thompson, and the administrator, Leonardo Brown. Your assistance and support was invaluable.

A special thanks to Pat Cannon and Jeffrey Woods for their support, encouragement, prayers, and for their invaluable mentoring.

A special thanks for the Computer Science and Engineering staff, especially Sri Satti, Randi Baldwin, for their help to do many paper works and for their guidance.

ABSTRACT

Malicious insiders increasingly affect organizations by leaking classified data to unauthorized entities. Detecting insiders' misuses in computer systems is a challenging problem. In this dissertation, we propose two approaches to detect such threats: a probabilistic graphical model-based approach and a deep learning-based approach. We investigate the logs of computer-based activities to discover patterns of misuse. We model user's behaviors as sequences of computer-based events.

For our probabilistic graphical model-based approach, we propose an unsupervised model for insider's misuse detection. That is, we develop Stochastic Gradient Descent method to learn Hidden Markov Models (SGD-HMM) with the goal of analyzing user log data. We propose the use of varying granularity levels to represent users' log data: Session-based, Day-based, and Week-based. A user's normal behavior is modeled using SGD-HMM. The model is used to detect any deviation from the normal behavior. We also propose a Sliding Window Technique (SWT) to identify malicious activity by considering the near history of the user's activities. We evaluate the experimental results in terms of Receiver Operating Characteristic (ROC). The area under the curve (AUC) represents the model's performance with respect to the separability of the normal and abnormal behaviors. The higher the AUC scores, the better the model's performance. Combining SGD-HMM with SWT resulted in AUC values between 0.81 and 0.9 based on the window size. Our solution is superior to the solutions presented by other researchers.

For our deep learning-based approach, we propose a supervised model for insider's misuse detection. Our solution is based on using natural language processing with deep learning. We examine textual event logs to investigate the semantic meaning behind a user's

behavior. The proposed approaches consist of character embeddings and deep learning networks that involve Convolutional Neural Network (CNN) and Long Short-Term Memory (LSTM). We develop three deep-learning models: CNN, LSTM, and CNN-LSTM. We run a 10-fold subject-independent cross-validation procedure to evaluate the developed models. Our deep learning-based approach shows promising behavior. The first model, CNN, presents a good performance of classifying normal samples with an AUC score of 0.85, false-negative rate of 29%, and false-positive rate of 26%. The second model, LSTM, shows the best performance of detecting malicious samples with an AUC score of 0.873, false-negative rate of 0%, and false-positive rate of 37%. The third model, CNN-LSTM, presents a moderate behavior of detecting both normal and insider samples with an AUC score of 0.862, false-negative rate 16%, and 17% false-positive rate. Moreover, we use our proposed approach to investigate networks with deeper and wider structures. For this, we study the impact of increasing the number of CNN or LSTM layers, nodes per layer, and both of them at the same time on the model performance.

Our results indicate that machine learning approaches can be effectively deployed to detect insiders' misuse. However, it is difficult to obtain labeled data. Furthermore, the high presence of normal behavior and limited misuse activities create a highly unbalanced data set. This impacts the performance of our models.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

An insider threat, defined as a damaging act practiced by an employee, is an emerging security threat against today's businesses. There are several reasons for an employee to become a malicious insider. For instance, some users misuse organizational resources because they either are unsatisfied with their work or they are looking for monetary gains [90].

Breach Level Index [28] is public information about data breaches collected and distributed by Gemalto. Breach Level Index asserts that around 40% of data leakage attacks are due to insiders' misuse. The data leakages are scored according to their importance. The risk scores of malicious insider threats range from 1 to 10 where 10 is the highest threat. These scores are highest in the USA and China with scores of 9.4 and 9.1, respectively. Recent studies [1, 27, 40, 14, 68] show that the insider threat rate has increased by 9% compared to 2015. The mean time to detect insider misuse is 50 days [18, 14, 68]. Insiders typically hide their malicious behaviors. This, coupled with a large variety of potential misuse, makes insider threat detection one of the most challenging problems in cybersecurity.

There are several solutions proposed to deal with insiders' misuse. User activities representing misuse are defined as low-frequency actions [27]. Researchers compare high-frequency actions to low-frequency ones to identify the unusual behaviors (see Chapter 2 for more details). Event logs, computer-based or network-based, have been extensively used to develop user profiles. For example, event logs can be used to monitor frequencies of actions by a user within a specific time unit.

Most of the insider detection systems use event logs to detect insider misuse [66, 87, 90,

99, 100]. There are two main approaches for log analysis: unsupervised or supervised [44].

Supervised learning algorithms require that the input data items are labeled with the appropriate category name. The labeled data items are used to train and evaluate the supervised learning-based algorithms to make predictions. These algorithms include, but are not limited to, linear and logistic regression, multi-class classification, and support vector machines. For example, in the context of insiders, a classification algorithm would learn how to identify malicious behaviors after being trained on logs of known insider's misuse activities that represent abnormal behaviors.

The main advantages of supervised approaches include high-accuracy and a predefined set of categories. On the other hand, the main disadvantages include the risk of overtraining (or overfitting) the classifiers, and high computation time.

Unsupervised machine learning algorithms aim to discover new knowledge from data, i.e., "knowledge discovery" [60]. One type of unsupervised learning is to use Hidden Markov Model (HMM). HMM handles sequence-based data samples. Thus, the sequence-based normal samples can be used to train HMM to create normal base lines. Then, the samples with low likelihood can be distinguished as abnormal behaviors.

Unsupervised approaches are computationally fast. Also, it is easier to get unlabeled data than labeled data. On the other hand, the main limitation of unsupervised models is that it is hard to identify the features to be used for modeling.

Pre-processing of event logs is crucial to ensure correct misuse detection. There are two main approaches proposed for event logs pre-processing: sequence- and tabular-based approaches. Sequence-based approaches aim to represent user events as a sequence of actions [66, 73]. The sequence-based approaches preserve not only the user actions but also the relationships among these actions. Tabular-form approaches aggregate user actions over time (e.g., day or month) and present the aggregated values in a tabular form [90, 87]. In this work, we use the sequence-based pre-processing approach.

The closest research to our work to detect insider misuse were presented by Rashid

et al. [73] and Yuan et al. [100]. Rashid et al. [73] proposed an approach using Hidden Markov Model (HMM) to profile user normal behaviors. Their research used week-based sequences of computer logs. User profiles are used to distinguish between the normal and abnormal behaviors. A limitation of their work is that week-based sampling is too long a time to identify suspicious behaviors allowing insiders to misuse valuable resources. Moreover, insiders may hide malicious behaviors among several sessions to fool anomaly detection systems. Yuan et al. [100] provide a deep learning-based approach to identify insiders misuse. Their work consists of two layers: LSTM for feature extracting and CNN for feature modeling. They used one-hot encoding to represent computer-based events. One-hot encoding leads to the "curse of dimensionality" because it creates a new dimension for each unique event [32]. Thus, the one-hot encoding requires large space and large memory. Moreover, this representation does not consider the semantic meaning of the event. Generally, the computer-based events are textual events that could involve access logs, device, email, website, and file logs etc. Thus, it is crucial to consider Natural Language Processing (NLP) to model these events. In our work, we address the above limitations.

1.1 Research Purpose

The main purpose of this work is to detect insider misuse. For this, we build computer log-based models to identify misuse of organizational data by an insider.

The methodology of this research is developed based on data sets of Insiders Detection Tool from Carnegie Mellon University (CMU). These data sets provide computer-based event logs of five different domains: http, email, thumb drive device, used files, and login/logout events. The insider data sets present the information of 1000 users collected over 11 months.

We present two methods to pre-process large data sets so that insider analysis is amenable to a typical desktop computer. The presented work try to fill the research gaps, as highlighted in the previous section, by developing machine learning detection models based on

a sequential sampling of the data. The contributions of this work can be summarized as follows:

1. Pre-processing computer-based logs of multiple domains as session-based sequences.
2. Developing unsupervised detection systems.
3. Developing supervised sentiment-based detection systems.

1.2 Research Objectives

The specific objectives of our work are:

1. Improving the detection accuracy by pre-processing computer event logs as session-based sequences, see task one for more details.
2. Reducing the resource needs of the pre-processing systems, i.e., limited size RAM, see task one for more details.
3. Improving ROC curve measured accuracy by using Sliding Window Technique (SWT) with HMM model, see task two for more details.
4. Autonomously extracting features that represent user behaviors by using deep Convolutional Neural Network (CNN) , see task three for more details.
5. Incorporating semantic meaning of user behaviors to improve detection accuracy. We present deep sentiment-based models that are able to build relations among user activities and extract the semantic meaning of user behaviors, see task three for more details.
6. Studying the changing effects of the hyper-parameters and network structures on the performance of deep learning-based models, see task four for more details.

1.3 RESEARCH TASKS

This dissertation presents our research findings by developing the following approaches:

1. Pre-processing approach: pre-process the unstructured computer-based log data to generate sequence-based data samples.

   We choose a sequential-base analysis approach to develop our detection systems because this approach reflects the actual behavior of insiders, who perform a series of actions such as logging in, downloading data, and copying to thumb drives. We are, as humans, doing our daily activities chronologically, as a sequence. Sequence-based approach preserves the semantic meaning of the actions and the relationship flow among the actions.

   We develop two pre-processing approaches to generate data samples:

   - Session-based sequences of encoded action events: the goal of this approach is to develop a model that discovers the frequency of activity patterns of a user based on a sequence of encoded computer event logs.
   - Session-based sequences of textual action events: the aim of this approach is to develop a model that identifies the relations among a sequence of computer-based actions.
   - Working with low size RAM: both of the developed preprocessing approaches are able to work with low size RAM. These approaches support researchers who are working on big data analysis and have limited resources.

2. Developing an unsupervised cybersecurity system to model normal behaviors and detect the abnormal ones. Our sub tasks to develop the unsupervised system are as follows:

- Developing a stochastic gradient-descent-learning based Hidden Markov Model (SGD-HMM) to process the encoded sequence data samples. SGD-HMM model is able to statistically find the sequence-based activity patterns.
- Developing a Sliding Window Technique that improve the detect of malicious behaviors that are distributed over a long-time period. For instance, this technique is efficient when a malicious user is trying to spread and hide his/her misuse behaviors through several time units to avoid being detected by security systems.
- Evaluating the performance of SGD-HMM model in terms of Receiver Operating Characteristic curves (or ROC curves).

3. Developing supervised deep learning models to process the textual sequence-based data samples. First, character embedding technique is used to map the sequence of action events as vectors. Then, three deep learning approaches are used to process these vectors. Our sub task can be summarized as follows:

   - Developing Long Short Term Memory (LSTM) model: We aim to model the sequence of embedded features. LSTMs have a feedback connections that consider the effect of the previous feature vector on the current feature vector.
   - Developing Convolution Neural Network (CNN) model: We aim to extraxt the most important features using different size filters, called "receptive fields." The convolutional layer reduces the model complexity by reducing the input dimensions.
   - Developing CNN-LSTM model: We aim to combine the CNN and LSTM approaches to model the sequence of extracted features.
   - Evaluating the performance of deep-based models in terms of Precision, Recall, F1 score, Accuracy, and Receiver Operating Characteristic curves (or ROC curves) with Area Under the carve (AUC).

4. Studying the effects of changing deep-based hyper-parameters on the performance of deep learning models. We aim to do the following:

    - Studying the impact of increasing the number of CNN or LSTM layers.
    - Studying the effects of increasing the number of nodes per layer.
    - Examining the effect of changing the number of epochs.

1.4 Research Contribution

In this dissertation, we developed the following insider's misuse detection systems :

1. Unsupervised: SGD-HMM model with Sliding Window Technique (SWT) to predict users' misuse behavior based on the users' activities.
2. Supervised: Deep learning-based models using Natural Language Processing (NLP) to distinguish between normal and malicious behaviors based on the context of the user activities.
3. Twelve deep learning architectures to study the effect of the number of layers and the number of hidden units per layer on the model's detection performance.

Other contributions include :

1. Preprocessing: Developed a preprocessing approach that does not need a large memory size.
2. Propose a way of using the augmentation algorithm (SMOT) and the fit function in Keras efficiently with unbalanced data.

1.5 Dissertation Outlines

The structure of the remaining sections are as follows:

1. Chapter 2 shows related works on insider threat detection systems.

2. Chapter 3 presents background information about: Hidden Markov Model (HMM), Artificial Neural Network (ANN), Convolutional Neural Network (CNN), Recurrent Neural Network (RNN), Long Short Term Memory (LSTM), and Natural Language Processing (NLP).

3. Chapter 4 shows our SGD-HMM model to detect insiders.

4. Chapter 5 presents our Natural Language Processing with deep learning approaches for insiders' threat detection.

5. Chapter 6 investigates networks with deeper and wider structures than the one we used in Chapter 5.

6. Chapter 7 offers conclusions and suggestions for future work.

CHAPTER 2

RELATED WORKS

Designing an efficient and scalable framework for monitoring and detecting malicious insiders is a significant research interest. Many research studies have been investigating and analyzing the problem, see [71] for an overview. This chapter will survey the techniques and challenges of detecting insider's misuse. We begin with studies that review psychological aspects that may lead to insiders' misuse behaviors. Then, we continue to review the works that analyze computer-based logs with the intention of detecting misuse. Our survey focuses on anomaly-based approaches. We categorize these works to unsupervised approaches, sequence-based, deep learning, semantic and network-based approaches. Also, we provide a brief discussion of the deception methods of insiders.

The works presented in [6, 35, 49, 55, 56, 74, 77] review technical, psychological and social perspectives of insider threat problem.

2.1 PSYCHOLOGICAL STUDIES

Bishop et al. [71] examined psychological signs to determine which users show the most considerable risk of misbehaving. These signs are categorized as follows, from least to highest in regard to significance: *dependability, absenteeism, self-center, personal issue, confrontational, stress, performance issue, disregard for authority, disengagement, anger management, accepting feedback, disgruntled*. Therefore, these indicators serve as signals of possible future wrongdoing.

Another study by Schultz in [80] defined five behavioral indicators that are predictive of an insider. These indicators include: personality traits such as introversion, deliberate

markers such as engaging in deviant behavior online, meaningful errors such as making mistakes, preparatory behavior such as using a range of system-level commands, correlated usage patterns such as a user exhibiting a particular behavior on multiple sub-networks or subsystems which separately do not show a suspicious pattern, but collectively do show a suspicious pattern, and verbal behavior in which hateful language is used.

The works in [20, 57, 95] aim to understand how various corporate factors can create disgruntlement among employees. Disgruntled employees are frequently mentioned as potential insider threats. According to Cooper et al. [46], behaviors can be measured by three essential properties: repeatability, temporal extent, and temporal locus. Repeatability refers to how behavior can be counted or how it can repeatedly happen within time. Temporal extent refers to how much time a behavior needs to occur. Temporal locus refers the point in time the behavior occurs.

Psychological studies provide useful information about the progression of an employee to become an insider and understanding the human elements of that process. However, these personality traits and this progression are difficult to observe in the computer system. Therefore, researchers use computer-based logs analysis to see if they can find indicators for anomalous misuses.

2.2 Computer Activity-based Anomaly Detection

Computer-based analysis uses data collected from the host events to detect anomalous behavior. These events range from applications such as emails, web-based applications, host-based applications or files, web sites, keystroke/mouse movements, etc. These data sources can be used to reflect a host behavior or user interaction behavior with the host [55].

A unique aspect of insiders' misuse is that there is no uniform model that describes how somebody progresses to become an insider. So, anomaly detection methods seem to be promising to warn us if there is a change in the pattern of the user behaviors.

There are anomaly detection-based systems that have limitations as well [15, 67]. For

example, if an employee changes his/her job's activities, then we will detect that as an anomaly activity. Whether it is a progression toward becoming an insider or it is a changing job function, it needs to be evaluated.

In this section, we present several computer-based anomaly detection systems that were proposed to detect insiders' misuse. A comprehensive survey of the research is provided in [15, 71], where adopted techniques are grouped into different categories based on the underlying approach.

In this section, we organize anomaly detection systems into the following groups: unsupervised, sequence log-based user profiling, deep-learning-based, and sentiment analysis in insider detection systems.

2.2.1 Unsupervised Learning-based Studies in Insider Detection Systems

Unsupervised analyses aim to arrange a collection of data samples into clusters. This approach is useful when no knowledge is available about data items to assign them into predefined classes. Clustering is usually performed to find patterns in unlabeled data [67]. Samples within the same cluster are more similar to each other than the items in the other clusters [34]. A classic approach to clustering is a K-means approach.

Chen et al. [17] introduced the community anomaly detection system (CADS). CADS is a computer-access based detection system that involves two tasks: pattern extraction and anomaly detection. Object/subject access frequencies were used to build a feature matrix. The matrix was used to characterize the distance value between the sets of patients accessed by each pair of users (e.g., patients' records are typically viewed by healthcare providers). Two methods were used: singular value decomposition [70] to infer communities, and KNN [54] to establish sets of nearest neighbors. The KNN method was used to determine if users have deviated from the behavior of existing communities. Gavai et al. [26] compared a supervised approach with an unsupervised approach using the Isolation Forest method at the task of identifying insider threats from system logs. They also aggregate information

to extract features that model user behaviors.

Young et al. [99] and Senator et al. [87] presented ensemble based unsupervised technique and used structural and semantic information of an organization. The proposed model uses the scores from multiple indicators that combine predictions from multiple classifiers. Besides these studies, Schubert et al. [31] and Balzer [103] presented ensemble-based techniques that combine multiple unsupervised anomaly detection algorithms to boost their joint anomaly detection performance. Since unsupervised anomaly detection does not rely on labeled data, this task is very challenging and often restricted to simple combinations [31].

These works are based on using statistical features that give insight about user behaviors. However, they do not indicate the semantic meaning of user actions. The presented approaches do not have a knowledge base about user actions or the relation among these actions. In this work, we aim to fill this research gap by developing techniques and systems that are semantically meaningful toward detecting malicious behavior.

2.2.2 Sequence Log-based User Profiling

Once the insider attack has occurred, it is useful to investigate the intent of the insider attack based on audit source [77]. In computer user profiling, various audit sources can be used to obtain information such as command line, system calls, database/file access, organization policy management rules, and compliance logs [22]. Most of the presented studies use statistical feature analysis such as the frequency of events, the duration of events, the co-occurrence of multiple events combined through logical operators, and the sequence or transition of events.

Ju and Vardi in [47] presented a hybrid high-order Markov chain model. The goal of the work was to identify a signature behavior for a particular user based on the command sequences that the user executed. Szymanski and Zhang [84] proposed recursively mining the sequence of instructions by finding common patterns, encoding them with unique sym-

bols, and rewriting the sequence using the new coding. Dash et al. [7] and Li et al. in [53] also created user profiles from groups of sequences commands. In [7], 13 temporal features were used to check the consistency of patterns of commands within a given temporal sequence. Probabilities were calculated for movements of commands within a sequence in a predefined reordering between commands.

Jha et al. [45] presented a statistical anomaly detection algorithm that has the potential of handling mixtures of traces from several users by using mixtures of Markov chains. The technique was compared to Hidden Markov Models (HMMs). Rashid et al. [73] claimed to be the first to adopt the Hidden Markov Model to the domain of insiders threats detection. They proposed to use the concept of "moment of inertia" to improve the accuracy of results. Even though sequence log-based studies were profiling normal user behavior and used them as baselines to distinguish the abnormal behavior, insiders may hide his/her behaviors among data samples to trap anomaly detection systems. In this work, we propose to use a sliding window technique with HMM that analyzes multiple data samples at once, which helps to see hidden behaviors, for more details see Chapter 4.

2.2.3 Deep learning Studies in Insider Detection Systems

Deep learning (DL) is a part of machine learning based algorithms that attempt to model high-level abstractions in data. Deep learning neural networks (DNN) show to be effective mechanisms in predictive analytic because they can model data that includes non-linear properties. In this section, we present several studies that applied deep learning techniques to develop their detection systems [10, 69, 90, 100].

Yuan et al. [100] presented an insider detection system using DL approaches. They used an LSTM-CNN framework to find user's anomalous behavior. The work consists of two parts: feature extracting and feature modeling. Long Short-Term Memory (LSTM) model was used to extract features from sequence-based data samples. A sequence of actions was mapped to a sequence of numbers. Then, one-hot encoding was used with each

number to provide a new representation of the input sample. Then, an LSTM model was trained with one-hot encoded samples to be used later to extract feature vectors. Finally, feature vectors were modeled using Convolutional Neural Network (CNN). Deep-learning techniques were applied in the presented work. However, the used encoding approach cannot hold the semantic meaning of insider actions in depth. The one hot encoding method encodes each event in one vector. For example, the textual event such as emails or web sites contents will be encoded as one vector. Therefore, it is hard to get enough information from the presented data samples. Tuor et al. [90] presented an unsupervised deep learning auto-encoder to learn which patterns of network activity were common. They used RNN (Recurrent Neural Network) and DNN (Deep Neural Network) to extract feature from raw data, which then used for threat assessment in data streams. Their algorithm used aggregations of network activity over time to find anomalous patterns. The occurrence of selected events were counted and aggregated to so-called user days. The weekdays, not weekends, were analyzed with the aggregation operations. In this work, we do not aggregate the events to the so-called user days, but we instead analyze the events on their own. Also, both weekdays and weekends are analyzed. Moreover, deep learning techniques LSTM and CNN with Natural Language Processing (NLP) are combined to investigate the semantic meaning of the user activities.

2.3 Sentiment Analysis in Insider Detection Systems

Natural Language Processing (NLP) is a subfield of artificial intelligence that transforms textual language into a machine-based representation that is clear for computers to handle. NLP has three main applications: rule-based, probabilistic modeling, and deep learning approaches. In our work, NLP with deep learning approaches are used. Specifically, we utilize deep-based sentiment analysis. Sentiment analysis is a part of NLP methods used to infer various psychological or emotional indicators based on textual data. It shows exceptional performance compared to traditional methods [102, 39]. Survey studies of sentiment

analysis are presented in [101, 3, 59, 79].

Sentiment analysis has proven successful in a range of security systems [62, 64, 83]. The sentiment analysis could be performed by removing the more familiar words to focus on the low-frequency information as in the work presented by Park et al. [64]. Alternatively, it could be applied to a specific type of words when a combination of lexical and parser features are applied to detect offensive language [62].

Several works combined ontological semantic technology and natural language processing to detect insiders [21, 83, 86, 97]. Symonenko et al. [83] applied natural language processing to distinguish among intelligence analysts who had accessed information outside their interest. The work was based on building a domain-specific knowledge baseline by interviewing with analysts. Then, a clustering was used to separate analysts' behaviors. Yilmazel et al. [98] presented a work in intelligence community. Support Vector Machines were used to develop document classification approach to detect insider threat issues. One noticeable limitation of these works is that they need much preparing work to generate a domain-specific knowledge baseline. Also, the presented platforms are not general. They are applicable to one domain based on the acquired baseline. In our work, we analyzed various computer-based events such as file, email, and web URL to generalize our model. Moreover, we implement insider detection systems using state of the art deep learning techniques with natural language processing that extracts the anomalous features autonomously without human interaction.

2.4 Network Activity-based Studies

The daily operations of organizations are based on networked infrastructures that connect various devices across departments to facilitate data accessibility and the sharing of network resources. Defending such infrastructures from various cyber attacks and threats is important [23]. Here we briefly outline the network-based insider detection approaches.

Myers et al. [61] studied how web server log data could be used to distinguish malicious

insiders who look to misuse internal resources. Yen et al. [96] developed an unsupervised system called Beehive to identify anomalies in an enterprise environment including policy violations and malware distribution.

2.5 Deception Studies in Insider Detection

In this section we survey several deception studies of insider threat detection [48, 65, 76, 89]. Honey* is often used as an umbrella term for deception systems. The models of these systems are honeypots: decoy resources that are placed in a computing system to be penetrated and compromised by the intruders [85]. For example, Thompson et al. [65] present a content-based framework to detect insider anomalies in accessing documents and queries. Salem et al. [76] apply the machine learning techniques to identify the malicious intent in information gathering commands. Kaghazgaran et al. [48] proposed a model to consolidate honey permissions into role-based access control.

Araujo et al. in [5] proposed a honey patch approach, which are patches that offer an extra level of security by defeating an attackers' capability to decide whether their attacks have succeeded or failed. Honey patch responds to attempted exploits by redirecting the attacker's link to a separate decoy environment. Feeding disinformation to the attacker in the form of falsified data is also used in [13, 78]. The presented Honey-Patches approach is limited only to known patched vulnerabilities that reduce its applicability.

The honey approaches are used with people also by developing honey people. By adopting the honey pot approaches, information on nonexistent people is created and formulated as active actors in social media [91, 93]. The developed actors are built to seem realistic and have relationships with other people. Then, the interaction with these artificial personalities is monitored to evaluate the behavior. Based on the evaluation, an alarm can be issued.

CHAPTER 3

PRELIMINARIES

3.1 INTRODUCTION

This chapter highlights the background information and theoretical representation of the fields to which this work contributes. The presented information is mainly divided into two categories: Hidden Markov Model (HMM) and Deep Learning with Natural Language Processing systems.

3.2 HIDDEN MARKOV MODEL

Our goal is to model the user behaviors based on computer event logs. We need a model that has a robust representation of log data to model user actions. Generally, the user actions can be interpreted as a series of tasks. For example, a user starting his day by accessing the system, reading emails, surfing the internet, among other actions, and logging out of the system. We think that the sequence of actions reflect the weekly, and daily routines of user behaviors. Thus, we believe that Hidden Markov Model has the essential elements to satisfy these conditions.

Hidden Markov Model (HMM) is one of the common machine learning models. It consists of a finite set of states. The transition between states is controlled by a set of probabilities called transition probability matrix, or transition probability distribution. Observing an event at a particular state S is based on the probability distribution of the events at that state. The transitions between states are triggered according to the observed event or symbol. However, the state itself is invisible to the external observer. Thus, the model is

called Hidden Markov Model.

HMM can be summarized as a graphical statistical model that consists of three components: the initial probability vector, the transition matrix, and the observation matrix. Given an input sequence sample Y, we can describe HMM as follows [73, 72, 37, 43, 52]:

1. Observed symbols sequence $Y = (y_1, y_2, ..., y_T)$ with observed symbols m $\in$ {1, . . . ,M}. M is the total number of observed symbols. T is the time stamp of the observed symbol m.

2. Hidden state sequence $z = (z_1, z_2, ..., z_T)$ with hidden state s $\in$ {1, . . . ,S}. S is the total number of hidden states. T is the time stamp of the hidden state z.

3. Initial probability vector π the initial probability of hidden states S: $\pi_s = P(z_i = s); s \in 1, ..., S$.

4. Transition matrix A, the probability of moving from state i at time *t-1* to state j at time t:

$$P(z_t = s_i | z_{t-1} = s_j); i, j \in \{1, ..., S\}. \tag{3.1}$$

5. Emission matrix B, the probability of observing symbol (m) from hidden state (s):

$$\begin{aligned} &P(y_t = m | z_t = s); \\ &s \in \{1, ..., S\}, m \in \{1, ..., M\} \end{aligned} \tag{3.2}$$

In this work, we follow the first order assumption about HMM which states that the transition probability of the hidden state at a time t only depends on the hidden state at the previous time point $t - 1$:

$$P(z_t | z_{t-1}, ..., z_1) = P(z_t | z_{t-1}) \tag{3.3}$$

Also, the observation at a time t depends only on the current hidden state, not on the previous hidden states or observations:

$$P(y_t|y_{t-1}, ..., y_1, z_t, ..., z_1) = P(y_t|z_t) \tag{3.4}$$

In the domain of our work, we focus on answering the following question:
What are the parameters of our models that increase the probability of the sequences that represent normal user behaviors and decrease the chances of the anomalous behaviors sequences?

3.2.1 USES OF HMMS

There are three problems that can be solved if a system is constructed as HMM:

1. **Evaluation:** calculating the probability of an observed sequence given a HMM.
2. **Decoding:** calculating the most likely sequence of hidden states that probably generated an observed sequence.
3. **Learning:** determining HMM parameters: initial probability vector, transition probability matrix, and the symbols probability matrix given a sequence of observations.

HMM has two main advantages compared to other machine learning approaches, such as Support Vector Machine SVM:

1. HMM evaluates different length sequence-based data points.
2. HMM models temporal information among sequence symbols.

3.3 NATURAL LANGUAGE PROCESSING WITH DEEP LEARNING

This section presents the theoretical background on artificial neural networks that are used with the proposed approaches. Section 3.3.1 illustrates the concepts of artificial neurons

and the structure of fully connected deep networks known as feedforward neural networks. The remaining sections describe the advanced structures of deep learning networks, such as Convolution Neural Network (CNN) and Recurrent Neural Network (RNN). Also, some of the important regularization techniques used with deep learning will be explained.

3.3.1 Artificial Neural Networks

Human neurons are the cells that receive and transfer chemical and electrical signals along brain channels. The brain has neurons of different shapes and sizes. The variety of sizes and shapes specifies the function of each neuron: for example, storing memories, controlling body function, etc. Figure 3.1 illustrates the basic structure of a brain neuron. There are three main parts of the neuron: dendrites, axon, and axon terminals. Dendrites are responsible for conducting electrical messages after they receive stimulation to the neuron's body to function. An axon typically carries electrical pulses from the neuron's body to transmit information to different neurons through axon terminals. The axons are known as nerve fibers. Axon terminals are isolated from neighboring neurons by a small gap called a synapse. The signals are sent across a synapse.

Developing artificial neurons are inspired by human neurons. Generally, biological and artificial neurons receive input signals on multiple connections and only produce an output signal based on the level of stimulation, or when a weighted sum of the inputs exceeds a threshold value.

$$y(x) = \sigma\left(\sum_{k=0}^{k} w_k x_k + b\right) = \sigma(w^T x + b) \tag{3.5}$$

To know the behavior of each artificial neuron, a mathematical representation of a single neuron network, known as a Perceptron, is shown in Equation 3.5. The network consists of a single neuron with K input values, and a non-linear activation function σ. Also, the network is parametrized by a weight vector w and a bias b [92].

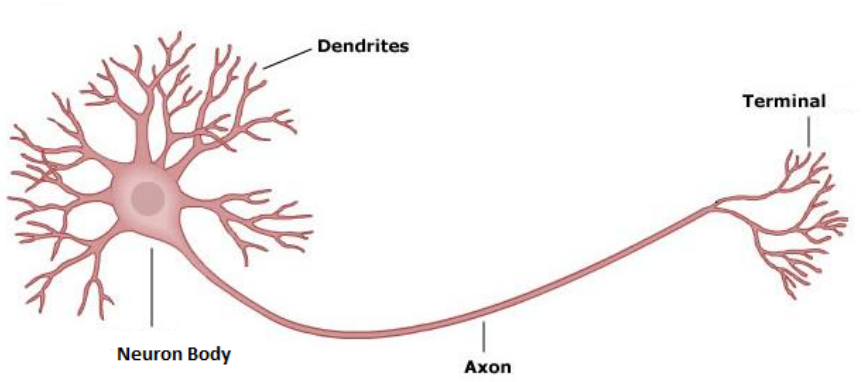


Figure 3.1: Structure of a Typical Neuron. The neuron has three main components: Dedrites, Axon, and Terminal. Dedrites are branches of a nerve cell that transfer the electrochemical stimulation obtained from other neural cells to the cell body. Axon transmits information to different neurons from the nerve cell body. Axon terminal makes connections with another nerve cell [94].

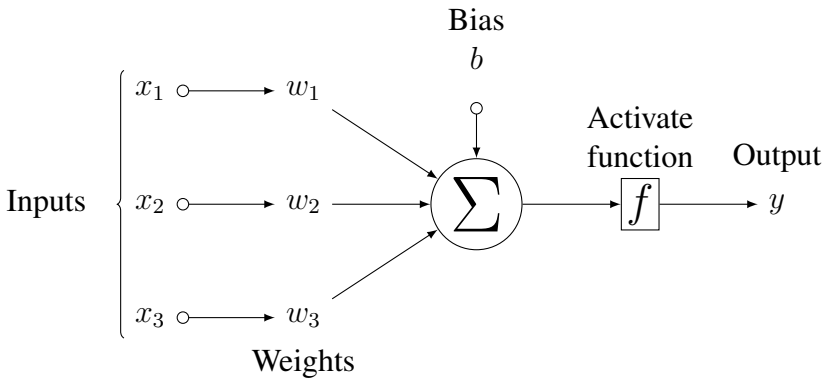


Figure 3.2: Artificial neuron, Equation 3.5, the neuron computes the weighted sum of an input vector $x = (x_1, x_2, x_3)^T$, adds a bias value, applies an activation function σ, and outputs the result.

Artificial, or feedforward, Neural Networks (ANN) is an optimization of a Perceptron as illustrated in Figure 3.3. ANNs receive a vector of numbers in the input layer, and pass it through a series of hidden layers. A hidden layer consists of a number of neurons, where each neuron is fully connected to all neurons in the previous and next layers. In each layer, a neuron is totally independent from others neurons by having its own activation function, input and output connections. The last layer is the output layer where all neurons are fully-connected to the previous hidden layer. In classification problems, the output represents the probability score of each class [32].

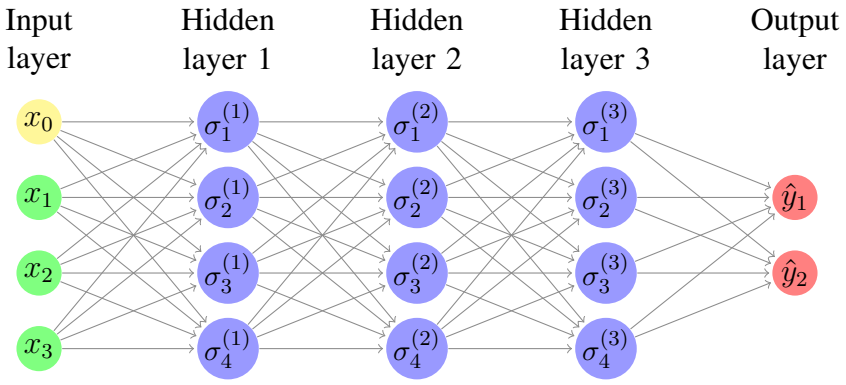


Figure 3.3: This is an example of a feedforward neural network that consists of an input layer *X*, three hidden layers of four neurons, and an output layer of two classes ($\hat{y}_1$ and $\hat{y}_2$). Every neuron at layer *i* is fully connected to all neurons in *i+1* layer to satisfy the feedforward condition. The output $\mathbb{R}_4 \Rightarrow \mathbb{R}_2$ is the composition of the hidden functions $\sigma(x) = \sigma(4)(\sigma(3)(\sigma(2)(\sigma(1)(x))))$.

According to the feedforward property, the outputs of all neurons of a layer i can be computed in parallel [32]. The number of neurons among layers may differ. All neurons in a layer i with $K(i)$ neurons act on an input vector $x(i-1)$. The output is a vector $x(i)$ that depicts a non-linear function $f^{(i)} : \mathbb{R}^{K^{(i-1)}} \Rightarrow \mathbb{R}^{K^{(i)}}$

$$x^i = f^i(x^{i-1}) \tag{3.6}$$

The function at each layer is illustrated as:

$$f^{(i)}(x) = \sigma^{(i)}(W^{(i)}x + b^{(i)}) \tag{3.7}$$

The parameters of each layer can be represented by the weight matrix and bias vector: $W^{(i)}$ and $b^{(i)}$ which are constructed based on individual weight w and bias b of neurons.

The overall network parameters θ can be represented based on layer parameters $W^{(i)}$ and $b^{(i)}$. Thus, the network function is defined in terms of θ and input vector x as follows:

$$y = f(x; \theta) \tag{3.8}$$

The results of the output layer represents the function of the whole network. The output of the ANN can be written as a composition of the network layers [32].

$$f(x;\theta) = (f(i) * f(i-1) * \cdots * f(1))(x) \tag{3.9}$$

Figure 3.3 shows an example of a feedforward neural network with three hidden layers, represented as a directed acyclic graph.

ANN is a static classifier with input vectors having a fixed length. However, there are many applications that use sequence pattern recognition approaches, such as speech recognition, machine translation, natural language understanding, video processing, and bio-information processing. In sequence recognition, the dimensionality of the input data points can be variable. To solve this problem, HMM can be used. It is one of the useful tools that deals with variable length data samples [81]. Moreover, several deep networks have been developed to deal with dynamic or sequential patterns such as Convolution Neural Network and Recurrent Neural Network.

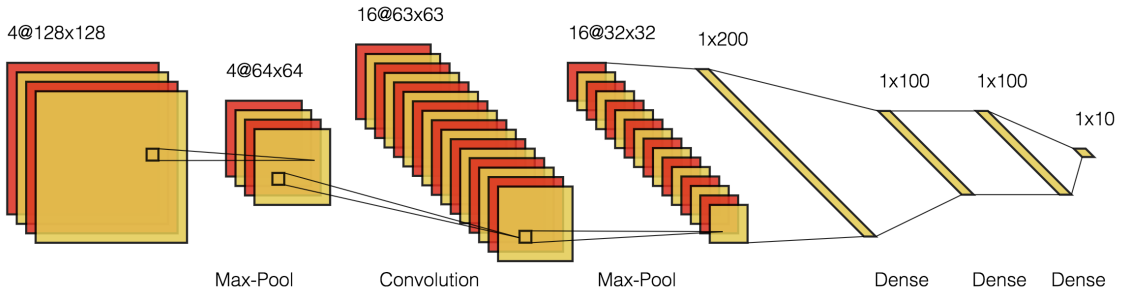


Figure 3.4: Example of LeNet style CNN network. The network consists of a single convolution layer, two max pooling layers, and the input and output layers. The first pooling layer downsamples the dimension of the input examples from 128x128 to 64x64 using 2x1 filter applied with a stride of 2. A convolution process is implemented with four filters to yield 16 feature maps. Then, a second pooling layer is used to reduce the computation complexity more and to generalize the model. The remaining components of the network are flattening the features maps layer followed by three dense layers. Finally, the output layer provides predictions for ten classes.

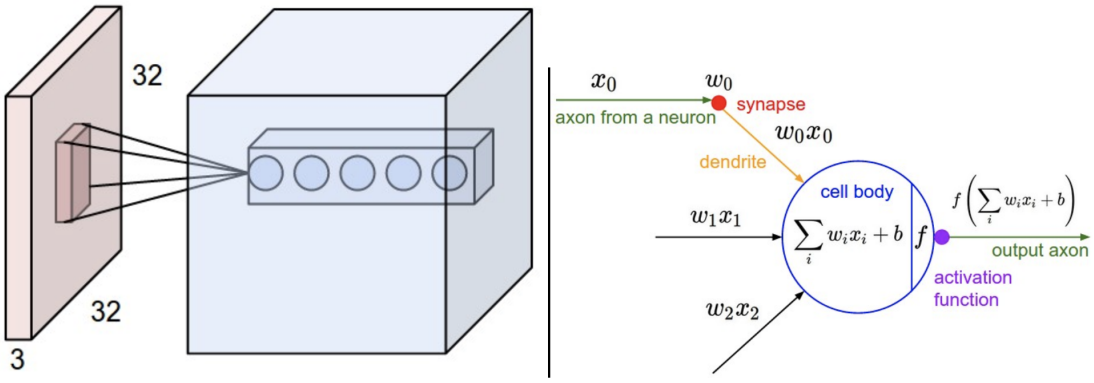


Figure 3.5: An example of five-neuron Convolutional layer inputs 32x32x3 CIFAR-10 image. Each neuron in the Convolutional layer is connected to a local region in the input volume (i.e., blush color); however, each one is connected to the full depth of the input sample (i.e., all color channels). All neurons work with the same receptive field in the input. On the right is a representation of a dot product performed by each neuron. The neurons compute a dot product of their weights with the receptive field region followed by a non-linear function [2].

3.4 Convolution Neural Network

Convolution Neural Network works by sliding a filter window over the input data points as shown in Figure 3.4. Then, it performs the convolution operation to find dot products between the entries of the filter and a local region of the input sample. This spatial region is a hyperparameter called the receptive field, and it matches the filter size. The sliding step is based on the filter size and shifting step stride. As in feedforword networks, each dot product operation is followed by a non-linear activation function. However, the connectivity of each neuron in the convolution process is now restricted to be local spatially as shown in Figure 3.5.

CNN is mainly used with images to extract features from the input samples. For example, if the CNN's input is an image of size $x \times y \times d$, the network applies a filter of size $x_f \times y_f \times d$, where $x_f \leq x, y_f \leq y$, on each spatial position $(x^{'}, y^{'})$ to result in an output of size $x_o \times y_o \times 1$, see Figure 3.6. The filter will specify a feature of the object regardless of the object's location in the input image.

Generally, a convolutional layer consists of multiple filters. The output of each filter

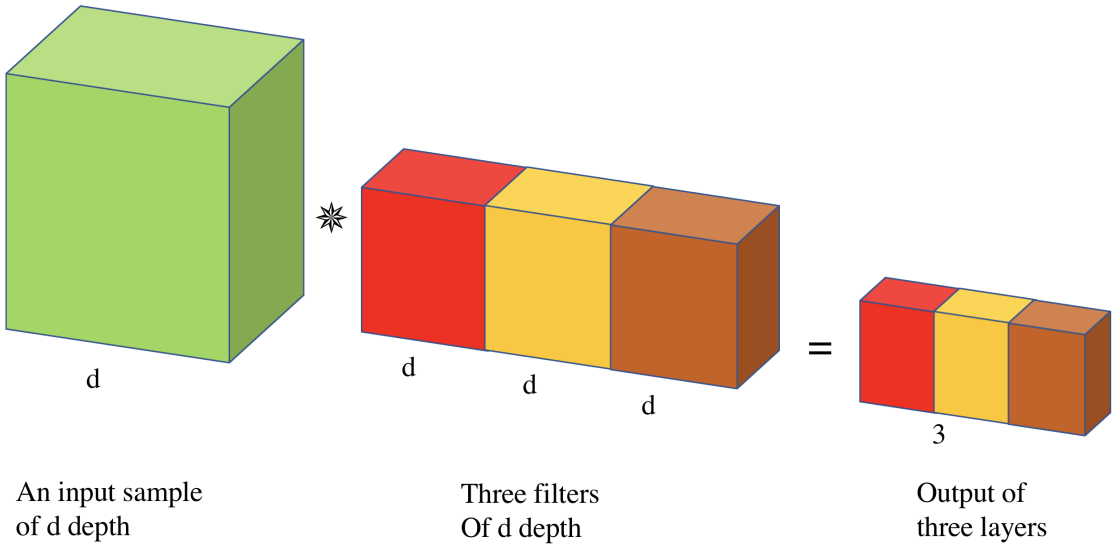


Figure 3.6: Example of a convolutional layer with three filters of depth d produce an output of depth 3 where each filter is applied on the whole depth d of the input volume to produce a sub output of depth 1.

$x_o \times y_o$, Equation 3.10, is stacked with other filters' output to form feature maps $x_o \times y_o \times n$, where n is the number of filters. A filter could have two or three dimensions, e.g., grayscale image, text data, or RGB image. An example of a CNN with four filters is shown in Figure 3.4. There are four data points of 64×64 volume and four filters. The convolution layer results in 16 stacked feature maps $16 \times 64 \times 64$. If the input sample has depth d, e.g., RGB image, the feature maps will include the depth as well $16 \times 64 \times 64 \times d$.

$$
\begin{aligned}
f(x; W, b) = & ReLU(Conv(x; W, b)) \\
& where: \\
& \textit{x is the input sample} \\
& \textit{W is the weights matrix} \\
& \textit{b is the bias vector}
\end{aligned}
\tag{3.10}
$$

Every layer of the CNN network has hyperparameters that can be tuned to improve the performance. Table 3.1 shows a summary of the CNN hyperparameters [2]. The spatial size of the output follows Equations 3.11, 3.12 where x, x_f, p, s_x represent an input sample,

Table 3.1: Overview of image-based CNN layers. Input size: $x \times y \times d$.

	Convolutional	**Pooling**	**Fully connected**
Hyperparameters	filter size xf , yf stride sx , sy padding px , py number of filters n	filter size x f , yf stride sx , sy	number of filters n
# of train. params.	(x f · yf · d +1) · n	none	(x · y · d +1) · n
Output size	$x \to \frac{x-x_f+2p_x}{s_x}+1$ $y \to \frac{y-y_f+2p_y}{s_y}+1$ $d \to n$	$x \to \frac{x-x_f}{s_x}+1$ $y \to \frac{y-y_f}{s_y}+1$ $d \to d$	$x \to 1$ $y \to 1$ $d \to n$

filter size, padding, and stride step, respectively:

$$x \to \frac{x - x_f + 2p_x}{s_x} + 1 \tag{3.11}$$

$$y \to \frac{y - y_f + 2p_y}{s_y} + 1 \tag{3.12}$$

3.4.1 POOLING LAYER

It is common to use pooling layers with CNN networks to reduce the dimensions of an input volume to decrease the number of parameters and computation in the network. It functions by applying a reduction operation on a small spatial neighborhood. In addition, the parameters' reduction helps to overcome or control the overfitting [32] .

The pooling layers act individually on every part of the input and resize it spatially, using the MAX operation. The typical examples are max pooling and average pooling. The most common form is a pooling layer with filters of size 2x2 applied with a stride of 2. These pooling operations downsample every depth slice in the input by 2 along both width and height as shown in Figure 3.7. The max process discards 75% of the entries of the pooling region. It takes a max over four numbers [2].

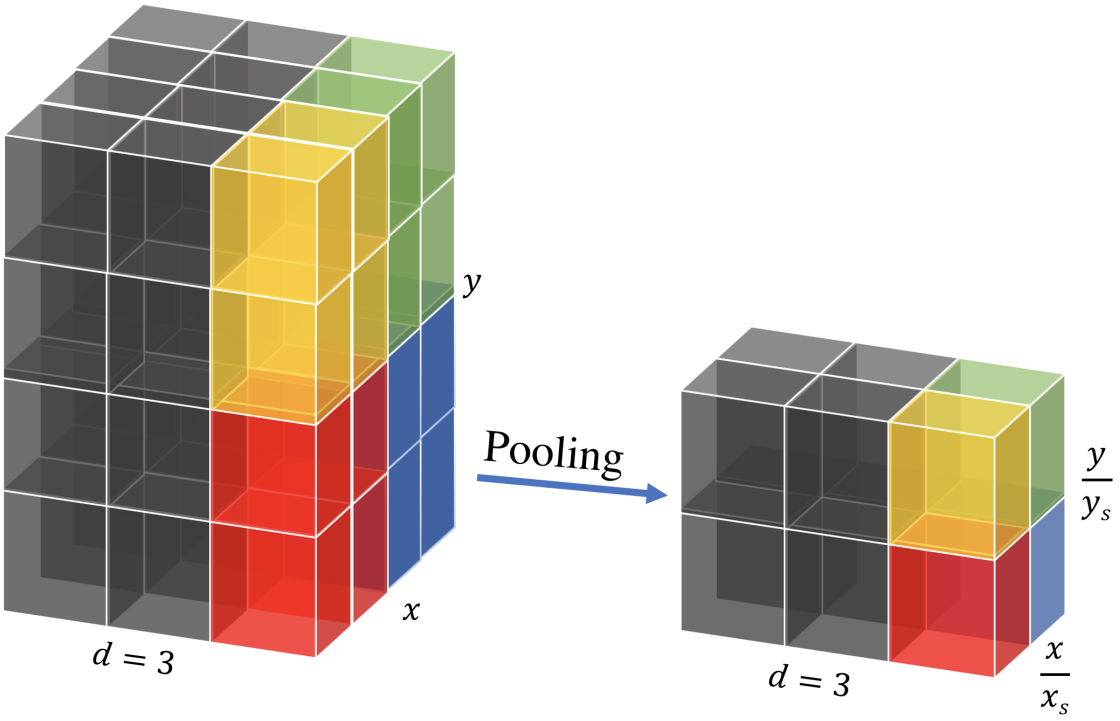


Figure 3.7: Example of pooling layer with stride 2 on a 2×2 window. Each of the d depth slices of the input volume is spatially reduced by applying a pooling operation on the elements inside the window. The pooling operation can, for example, compute a composite value, such as the average from the window's values, or select a value, e.g., by applying the maximum. As pooling is applied separately on each depth slice, the output volume has the same depth as the input volume.

3.4.2 Recurrent Networks

Humans are sequential thinkers. They do not begin their reasoning from scratch at every moment. For example, as a reader reads an article, he understands each word based on his understanding of preceding words. He does not start thinking from scratch at every word [63].

Traditional neural networks cannot retain the relation among successive events; Recurrent Neural Networks address this issue. RNN is any network that contains a loop within its network connections. The loop represents the feedback signal from the network output to its input. Figure 3.8 illustrates a simple RNN network with a feedback loop. The network A takes an input x_t and outputs a value h_t. A feedback connection sends information from

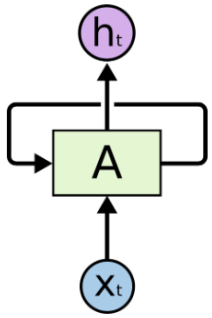


Figure 3.8: Recurrent Neural Networks with feedback loops [63].

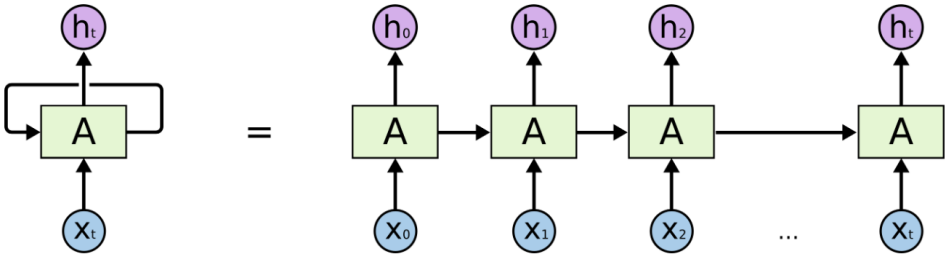


Figure 3.9: An unrolled recurrent neural network [63].

one round of the network to the next one.

The formation of the network is similar to that of a conventional feedforword network, but with one difference: it supports feedback links among hidden layers associated with a time delay. Figure 3.9 shows a unrolled version of the same RNN, A.

The information about the past can be retained through the loop connections of the network. The recurrent function helps to discover the temporal correlation among a sequence of events that are separated by a time step [75]. The inner state of the RNN will be updated while reading each input in the sequence sample. For instance, with a sequence input x of length τ denoted as $x^{(1)}, ..., x^{(\tau)}$, the updates of the Recurrent Neural Network's parameters can be represented as shown in Equation 3.13. RNN is suitable to use with sequential data, such as text or videos. On the contrary, the ANN and CNN networks produce a deterministic output for each input data point, independent from previous inputs or outputs [32].

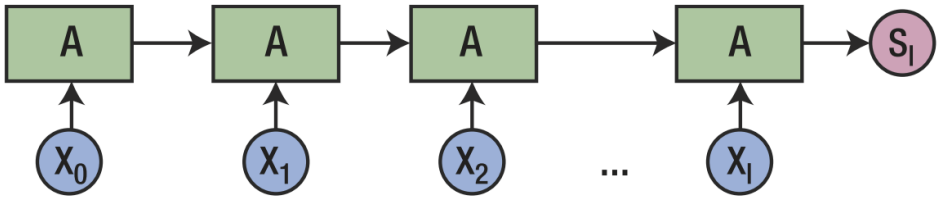


Figure 3.10: Encoding RNNs [33].

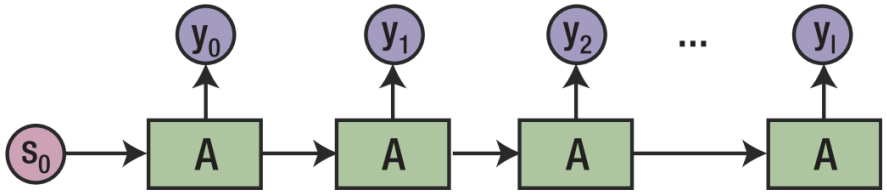


Figure 3.11: Generating RNNs [33].

$$h(t) = f(h(t-1), x(t); \theta) \tag{3.13}$$

There are various types of RNNs [33]:

1. **Encoding recurrent neural networks:** This set of RNNs takes an input of a sequence form and outputs an encoding style. This kind of network is an essential part of the encoding-decoding system used for language translation, Figure 3.10.

2. **Generating recurrent neural networks:** Such networks output a sequence of numbers or values, like words in a sentence, see Figure 3.11.

3. **General recurrent neural networks:** These networks are a combination of the preceding two types of RNNs. General RNNs (Figure 3.12) are used to produce sequences and, thus, are broadly applied in Natural Language Generation (NLG).

Even though the recurrent network is a simple and robust model, in practice, it is difficult to train properly. The main reasons behind this difficulty are the vanishing gradient and exploding gradient problems described in [8]. For example, consider a language model attempting to predict the next word based on a given sentence. If the model is trying to

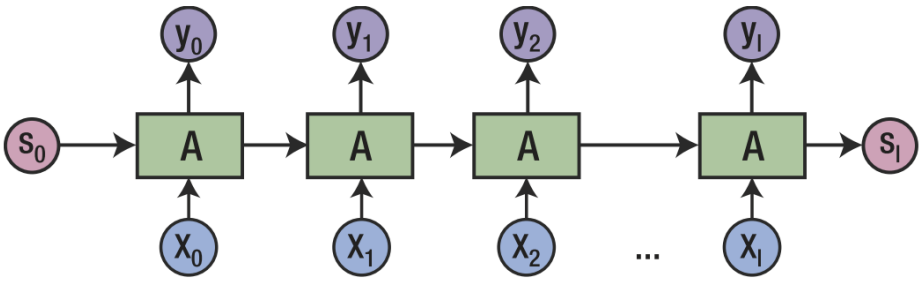


Figure 3.12: General RNNs [33].

predict the last word in “the clouds are in the sky,” the answer should be the word sky. The model uses the relevant information in the preceding words “the clouds are in the.” In such cases, the gap between the words is small. So, RNNs can learn to use the past information. Unfortunately, as that gap increases, RNNs are incabable of combining the information [63].

3.4.3 Long Short Term Memory networks

Long Short Term Memory networks (LSTMs) are a specific kind of RNN able to learn long-term dependencies. LSTMs were developed by Hochreiter & Schmidhuber [38] to avoid the long-term dependency problem. Remembering information for long periods is their default function.

All RNN networks are a series of a feedforward network. In usual RNNs, every single module has a simple structure, such as a single tanh layer [63], Figure 3.15. LSTMs also have a chain-like structure similar to RNNs. However, instead of one neural network, LSTM has four networks as described in 3.16 [63]. The interactions among the LSTM networks are based on two main components: cell state and controlling gates. LSTMs have three controlling gates:

1. **Cell state:** The cell state transfers information media like a conveyor belt. It transfers signals through the entire network chain as presented in Figure 3.13.

2. **Forget gate:** This gate decides what information to throw away from the cell state.

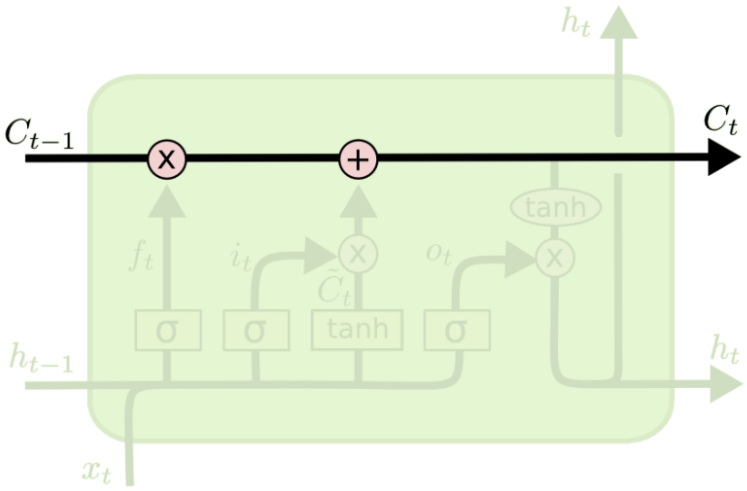


Figure 3.13: Cell State component of LSTM [63].

A sigmoid layer makes this decision. This layer processes h_{t-1} and x_t , and outputs a number between 0 and 1 for each number in the cell state C_{t-1}. A number 1 represents "keep the information" while a 0 represents "forget the information."

$$f_t = \sigma(W_f.[h_{t-1}, x_t] + b_f \tag{3.14}$$

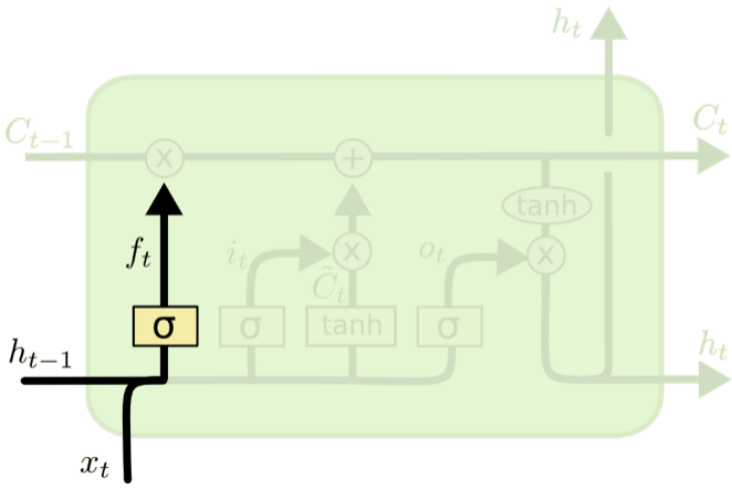


Figure 3.14: Forget gate of LSTM [63].

3. **Input gate:** The input gate decides what new information to store in the cell state, see Figure 3.17. First, the input gate, using sigmoid function, determines which values

to update. Then, a tanh layer generates a vector of new candidate values, $C_t^{\sim}$ 3.15. The last step is to combine the results of the sigmoid and tanh functions to produce an update to the cell state.

To discard the unwanted information, f_t is multiplied by the old cell state. Then, a piece of new candidate information, $i_t * C_t^{\sim}$, is added to yield new candidate values as described in Figure3.18 and Equation 3.16.

$$\begin{aligned} i_t &= \sigma(W_i.[h_{t-1}, x_t] + b_i) \\ C_t^{\sim} &= \tanh(W_c.[h_{t-1}, x_c] + b_i) \end{aligned} \tag{3.15}$$

$$C_t = f_t * C_{t-1} + i_t * C_t^{\sim} \tag{3.16}$$

4. **Output gate:** This gate outputs a filtered version of the cell state. The filter procedure consists of two main steps 3.19. First, a sigmoid layer is applied to decide the output part of the cell state. Second, a tanh function is used with the cell state to set the values between -1 and 1 and multiply it by the output of the sigmoid gate. This would control the output as desired[63].

$$\begin{aligned} o_t &= \sigma(W_o[h_{t-1}, x_1] + b_o) \\ h_t &= o_t * \tanh(C_t) \end{aligned} \tag{3.17}$$

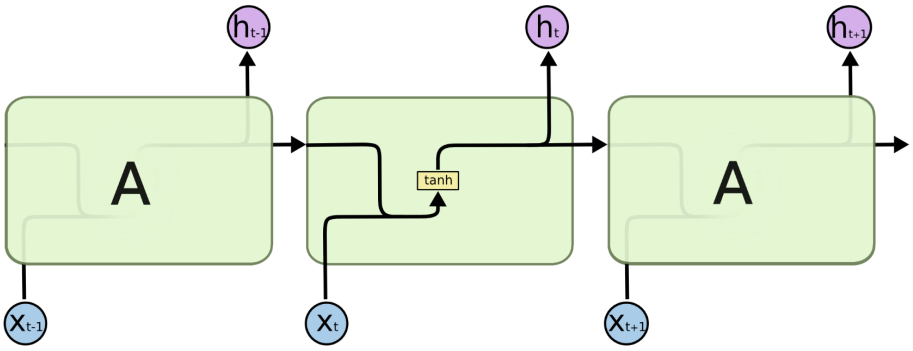


Figure 3.15: An example of a standard RNN contains a single layer [63].

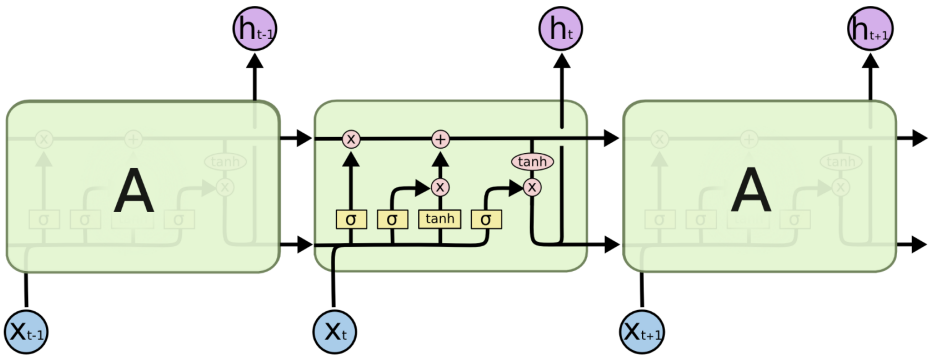


Figure 3.16: An example of a LSTM contains four interacting layers [63].

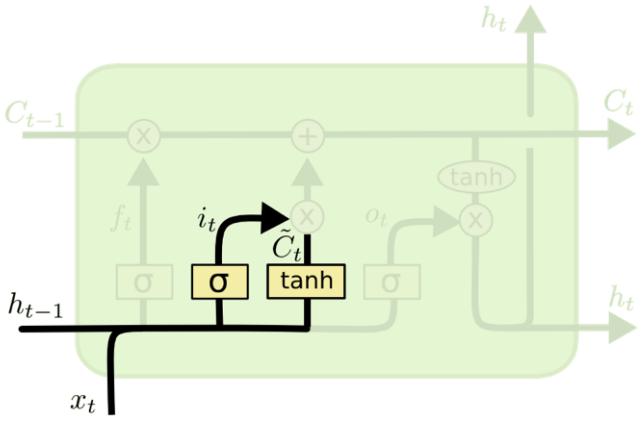


Figure 3.17: Example of input layer showing the use of sigmoid and tanh function s to determine what information to save [63].

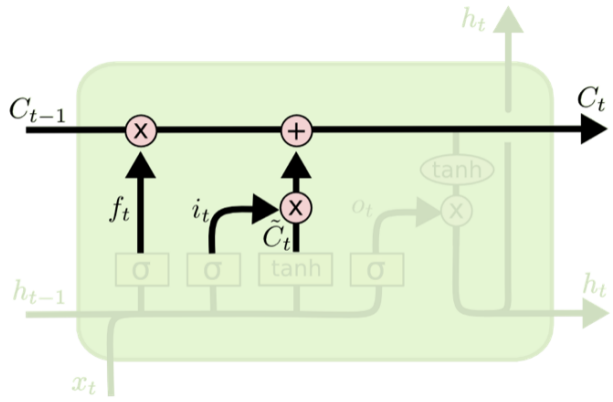


Figure 3.18: Example of using input gate layer to produce a new candidate values. The result of multiplication of sigmoid and tanh layer is added to cell state to decide the new candidate values [63].

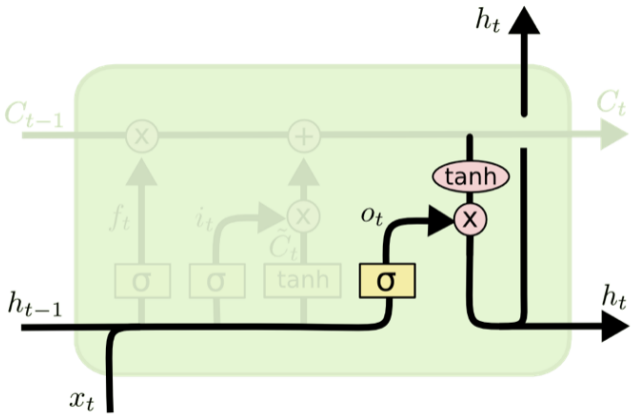


Figure 3.19: An output gate layer of a LSTM unit. It determines the output of the cell state using a sigmoid function and controls it using a tanh function [63].

3.5 Natural Language Processing

NLP is the bridge between the human language and the machines. It combines machine learning and linguistics to understand written or spoken languages. NLP is a way to represent, analyze, and interpret textual data, such as letters, words, and sentences, so that machines can understand them [9].

There are several applications of NLP with machine learning:

1. **Spam detection:** the Gmail algorithm distinguishes inbox emails and separates them into different categories, such as social and promotion tabs.

2. **Parts of speech (POS):** it stands for a process of identifing which words in a sentence are adverbs, adjectives, nouns, pronouns, verbs, and so on.

3. **Named entity recognition (NER):** It differentiates the meaning of a word. For example, the word "Jim" refers to a person, while the word "Apple" refers to an organization or fruit, and "2006" refers to a date.

4. **Machine translation:** This category involves language interpretation applications such as Google translate. The central role is to input a word or a phrase and convert it into several different languages.

5. **Machine conversations:** It is similar to the previous point, but instead of translating text to different text, it will translate a voice into a string of words. For instance, the Siri in Apple smart devices or Cortana in Windows operating systems are machine translation based applications. They receive voice signals and translate them into a logical text sequence and reply with something meaningful.

6. **Paraphrasing and summarization:** The NLP systems fall into this category when they take a document (a list of words) and produce a new textual material with fewer words.

7. **Sentiment analysis:** The NLP systems evaluate a document and give their answer as a negative or positive score. For example, if we want to assess the reviews of a restaurant, hotel or electronics devices store, we read all reviews, understand their meaning and give our scores. NLP systems are trained to understand the meaning and relationship between words in order to read and evaluate the documents.

3.6 Operation of Natural Language Processing

There are several pre-processing operations used with textual data to fit with machine learning algorithms, such as Naive Bayes Decision, or neural networks, etc. Machine learning

algorithms work with a vector of numbers. Thus, we present approaches to represent text as a vector of number to be used with machine learning approaches.

3.6.1 BAG OF WORDS

The Bag of Words (BOW) determines all the words in a document without considering the words' relation or frequency.

To pre-process the texts directly with Bag of Words, we specify which word is found in a document and which one is not. We can use this concept as a vector of numbers. For instance, if we want to convert the sentence, **"I like to eat pizza"** to a vector of numbers with the word dictionary involving the words: I, eat, like, drink, tomato, to, pizza, the vector representation will be as shown in Table 3.2:

Table 3.2: Bag of Words example.

I	**Eat**	**Like**	**Drink**	**Tomato**	**To**	**Pizza**
1	1	1	0	0	1	1

The first index of the vector presentation refers to the word "I", the second refers to "Eat", the third refers to "Like", and so on. Every word in a dictionary is assigned to an index in the vector. The number "1" or "0" is assigned to each index to indicate whether a word is in a document or not; "1" stands for YES, "0" stands for NO. If the same word appears more than once, the BOW vector is still the same. For example, the BOW vector of **"I like to eat pizza pizza"** is the same as Table 3.2. There is a similar technique used in machine learning to deal with categorical or indicator variables called "one-hot-encoding".

3.6.2 WORD FREQUENCY

The word frequency operation is similar to BOW, with one exception. The resulted vector is not just indicating the appearance of a word in a document; it also shows the frequency of a word, or how many times a word appears in a text. For example, the Word Frequency

vector of **"I like to eat pizza pizza"** presents the word "pizza" with the number 2, because it appears two times, as illustrated in Table 3.3.

When words have a high frequency in a document, the optimized operation TF-IDF (Term frequency - Inverse document frequency) differentiates between the low and high-frequency terms.

Table 3.3: Word frequency example.

I	**Eat**	**Like**	**Drink**	**Tomato**	**To**	**Pizza**
1	1	1	0	0	1	2

3.6.3 Term Frequency - Inverse Document Frequency

Generally, words without information (is, the, that, etc.) appear more frequently in a document. High frequency words have low impact on the semantic meaning of a document, while low impact words do not. To distinguish between them, two factors Term Frequency (TF) and Inverse Document Frequency (IDF) are calculated to assign a weight for each term in a document [58].

Applying TF with text results in every term t in document d is assigned a value based on the number of occurrence of the term in the document as presented in Equation 3.18.

$$TF(t) = \frac{(\text{\# of t appears in a document d})}{(\text{Total number of terms in the document d})} \tag{3.18}$$

With TF, all words are considered equally processed. However, There are words of little importance, such as "is", "of", etc. The goal is to scale down the high-frequency terms and scale up the low-frequency words. We can reach this goal by calculating IDF as described in Equation 3.19.

$$IDF(t) = Log\frac{\text{Total number of documents}}{\text{Number of documents with term t}} \tag{3.19}$$

Finally, TF-IDF is obtained by multiplying TF with IDF as presented in Equation 3.20

$$TF - IDF_{t,d} = TF_{t,d} \times IDF_t \tag{3.20}$$

3.7 Natural Language Processing with Deep Learning

To use Natural Language Processing (NLP) with deep learning, we need to adopt one of the previous word operations to represent textual data samples. However, word to vector (word2vec) represents words or letters in a new way. Such applications learn word relationships effectively, as shown in the following textbook example:

"woman" + "king" - "man" = "queen"

By using a well-trained word2vec, it is possible to obtain and understand the relationships among words.

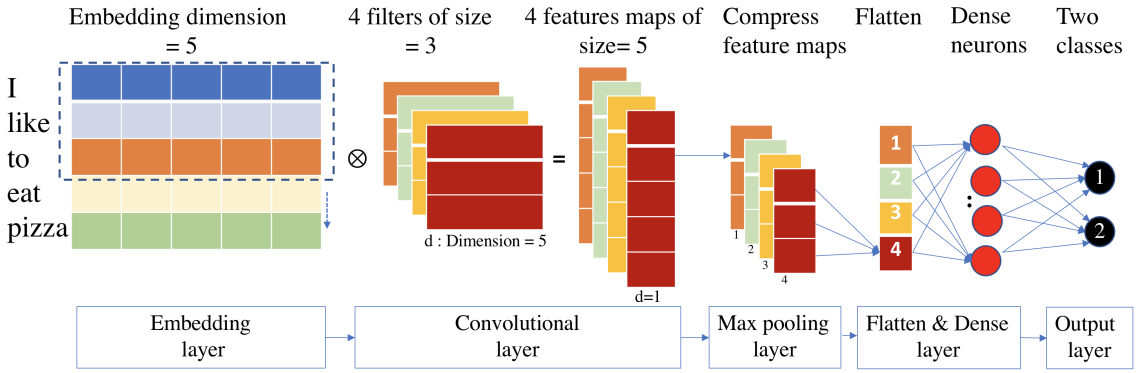


Figure 3.20: Example of NLP with a CNN. The network consists of a single dimension convolution layer that involves 4 filters of size 3. A convolution process is applied with padding and stride 1 on a 5×5 input volume. Each of the filters (in orange, green, yellow, and red) operates across the two dimensional matrix to yield four features maps, one of each filter.

3.7.1 Sentiment analysis using Deep Learning

Language sentiment analysis with modern deep learning techniques becomes more popular in many applications, such as customer feedback, restaurant reviews, etc. In this section, we

present a demo example of sentiment analysis using NLP with CNN. Figure 3.20 previews an NLP with CNN language model. The model consists of several components:

1. A single 1 dimension convolution layer that involves 4 filters of size 3. These filters are used to extract the tri-gram features between words.

2. A convolution process is applied with padding and stride 1 on a 5×5 input volume. Each of the filters (in orange, green, yellow, and red) operates across the two-dimensional matrix to yield four features maps, one of each filter.

3. The features maps are passed through the Max Pooling layer to reduce the complexity of the computations operations.

4. The compressed feature maps are flattened and passed through the fully connected layer.

5. Finally, the signals are passed to a classification layer to give a positive or negative score.

CHAPTER 4

PROBABILISTIC GRAPHICAL MODEL ON DETECTING INSIDERS: MODELING WITH SGD-HMM

This chapter presents a novel approach to detect malicious behaviors in computer systems. We propose the use of varying granularity levels to represent users' log data: Session-based, Day-based, and Week-based. A user's normal behavior is modeled using a stochastic-based Hidden Markov Model, HMM-SGD. The model is used to detect any deviation from the normal behavior. We also propose a Sliding Window Technique to identify malicious activity effectively by considering the recent history of user activity. We evaluated our results using Receiver Operating Characteristic curves (or ROC curves). Our evaluation shows that the results are superior to existing research by improving the detection ability and reducing the false positive rate. Combining sliding window technique with session-based system gives fast detection performance with (91) ROC value.

4.1 INTRODUCTION

Insiders' misuse of computer systems is a major concern for many organizations. Breach Level Index [28], public information of data breaches collected and distributed by Gemalto, asserts that around 40% of data leakage attacks are due to insiders' misuse. The data leakages are scored according to their important. The risk scores of malicious insider threats are the highest in USA and China: 9.4 and 9.1 respectively. Additionally, the recent studies in [27, 40, 14, 68] show that the insider threat rate has increased compared to 2015 which was 39%. The mean time to detect such malicious data breaches is 50

days [18, 14, 68]. Detecting insiders' misuse is one of the most difficult problems in Cyber Security.

There are several solutions proposed to deal with insider threat behaviors. Most of them define the suspicious behaviors as the low-frequency actions that are performed by a user. So, the unusual behaviors can be compared to high-frequency behaviors to predict the abnormality. The activities can be captured by tracing log data within a specific time unit. The actions' log data can be pre-processed such that it can be modeled using machine learning techniques [73]. However, none of these researches address the fact that a long time period is needed to detect malicious behaviors.

Processing and reshaping the data has a significant effect on the performance of the adapted models. For example, Gavai and Rolleston in [27] pre-processed the raw log data to generate several statistical features used to model user behaviors. Also, Rashid, Agrafiotis, and Nurse in [73] pre-processed their log data as a sequence of actions performed by a user during a week.

In this work, the raw data from five different domains, "Logon/Logoff," "Connect/ Disconnect," "Http," "Emails," and "Files," are pre-processed to generate new sequence data samples. Multiple domains show different aspects of user behaviors which would support our model to detect malicious behavior. The new data samples are generated according to the detection time unit as follows:

1. Session-based sequences
2. Day-based sequences
3. Week-based sequences

The session-based samples are low-level granularity samples. They have short length activity sequences compared to day-based samples, while the week-based samples have the most extensive activity sequence. Each one of the above representations indicates the detection time unit. For instance, a session-based sample means the user activities will be

evaluated after each session. In the same context, a day-based sample will be assessed at the completion of each work day and so on.

We propose an unsupervised detection approach to monitor user actions and detect the abnormal behaviors. A user's behavior is represented as a series of activities performed within the organizational environment. To identify the unusual sequence of actions, a stochastic gradient descent version of HMM, "HMM-SGD", is proposed to model the sequence of user activities. The new model has training flexibility because it contains four hyper-parameters. These hyper-parameters can be tuned to improve model convergence.

Our contribution in the presented work can be summarized as:

1. Processing the raw log data to be in session-based, day-based, and week-based sequences. Level granularity data samples help to discover the abnormal behaviours that are distributed over time.

2. Proposing a sliding window technique to consider the effect of the recent history of user activities on their current behavior.

3. Proposing the "HMM-SGD" to model the sequence data samples.

4.1.1 Training of Hidden Markov Model

This section illustrates how we train our HMM. As previously explained in Section 3.2, HMM has three parameters that need to be prepared: initial probability vector (π), transition matrix (A), and emission matrix (B).We use the Baum-Welch algorithm to train the parameters of our model. The Baum-Welch is an HMM context algorithm of the expectation maximization (EM) algorithm. Details of EM algorithm can be found in [11]. The training process can be set according to the structure of the adapted model. For example, Figure 4.1 illustrates a four-state structure HMM. We need to find the initial distribution of each of the four states and the transition distribution between them. Also, the distribution

of the observed symbols at each state should be determined as well. The list below shows how the model parameters are trained:

1. Initializing model parameters π, A, B with positive random numbers between 0 and 1, where:
   - (π) : The initial distribution of the states. The most probable state that the model will start with.
   - (A) : The initial distribution of the transitions between states.
   - (B) : The initial distribution of the observed symbols.
2. Baum-Welch algorithm is applied to learn HMM parameters. The details of the Baum-Welch algorithm are also presented in [72].
3. To make sure that there are no zeros within any of trained HMM parameters, we add a small number to each one of the parameters, followed by a scaling process to ensure the probability condition; all numbers in the symbols matrix add up to one. In addition to that, we use the scaled version of Hidden Markov Model, which also works on overcoming the resolution problems during the training process. Information about the scaled version of HMM is provided in [72].

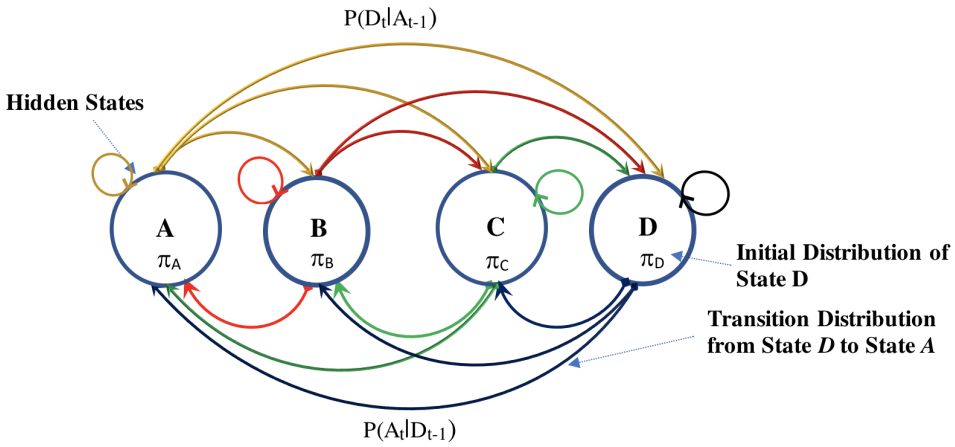


Figure 4.1: The structure of an insiders threat detection system with HMM.

The training process aims to find the model parameters that maximize the likelihood of the sequences that represent the user's normal behavior and minimizes the probability of the sequences that represent the anomalous behavior.

4.1.2 Training with Stochastic Gradient Techniques

As the second approach to model user behavior, we adapt a Hidden Markov Model with Stochastic Gradient Techniques (HMM-SGD) . The main difference in using HMM-SGD is the learning step. In the first method, we use the Baum Welch algorithm to train the model parameters While in this approach we use the stochastic gradient descent (SGD) algorithm to train the HMM. The gradient descent (GD) algorithm is the core algorithm of the training process in the Deep-Learning approaches (the deep neural network) and several others [51]. In this approach, we use the SGD method with SoftMax normality function to ensure the probability condition. According to our knowledge, we are the first who use HMM-SGD to solve the insider's threat attack problem.

4.1.3 Selection of GD

The learning methods are divided into two main categories: Stochastic-based and batch-based learning. For further explanation, we summarize them as follows:

1. Batch-based learning approaches: needs to process all the training data samples, insider action sequences, to update model parameters.

2. Stochastic based Learning approaches: each single sequence sample is used to update model parameters.

In the second approach, we adopt the stochastic base learning approach for several reasons [51]:

1. Stochastic-based learning is commonly faster than batch-based learning.

2. Most cases of stochastic-based learning leads to a better solution.

3. Since it is a sample base update, i.e., actions sequence, it is possible to track changes.

4.1.4 HMM-SGD LEARNING

Gradient Descent methods are the base of the most successful models, especially in deep learning systems [51]. These methods are used to learn parameters during a maximum number of iterations or when there is no change in model performance. The goal of using GD methods is to increase the likelihood of the input data samples, i.e., user activities sequences, given the model parameters. The training procedure works to fit the model parameters with the training dataset such that we can get a high likelihood of a new set of parameters. It is assumed that after scanning all iterations, the parameters will be updated in such a way that the model will converge with a high-objective value.

$$
\begin{aligned}
Objective(sequence_i) &= P(Q, O) \\
&= \pi_0(q_0) \prod_{t=1}^{T} P(q_t \mid q_{t-1}) \bullet P(o_t \mid q_t) \\
&\textit{where:} \\
&\textit{Q is hidden states sequence, } q_t \in \{q_1, ..., q_T\}, \\
&\textit{O is a sequence of the observed symbols,} \\
&o_t \in \{o_1, ..., o_T\} \\
&\pi_0 \textit{ is the initial states distribution} \\
&\textit{A is transition matrix: } A_{i,j} = pr(q_t = i | q_{t-1} = j) \\
&\textit{B is emission matrix: } B_{k=1}^{T} = pr(o_t = o_k | q_t = j)
\end{aligned}
\tag{4.1}
$$

The objective function is the joint distribution of the hidden state q at the time t and the observed symbols sequences $o_{i1}, ..., o_{it}$ given the model as described in Equation 4.1. We

train the model to get an objective value for the training sequences. The training samples represent the sequences of user actions within each session as described in section 4.5.1.

The essential formula of Gradient Descent is illustrated in Equation 4.2. The GD algorithm uses the chain rule to accomplish the training goal for all model parameters [88]. The context of HMM with gradient descent can be summarized as follows:

1. The term W(t) refers to any of the current parameters {π, B, A}.
2. W(t+1) presents the updated version of the model parameters.
3. To orientate the system learning process, we manipulate the learning rate parameter "μ" that changes the learning step during the training procedure.
4. The gradient term of the Equation 4.2 presents the derivation of the objective function with respect to the model parameters.

$$W(t+1) = W(t) - \mu * \frac{\partial Objective}{\partial W} \tag{4.2}$$

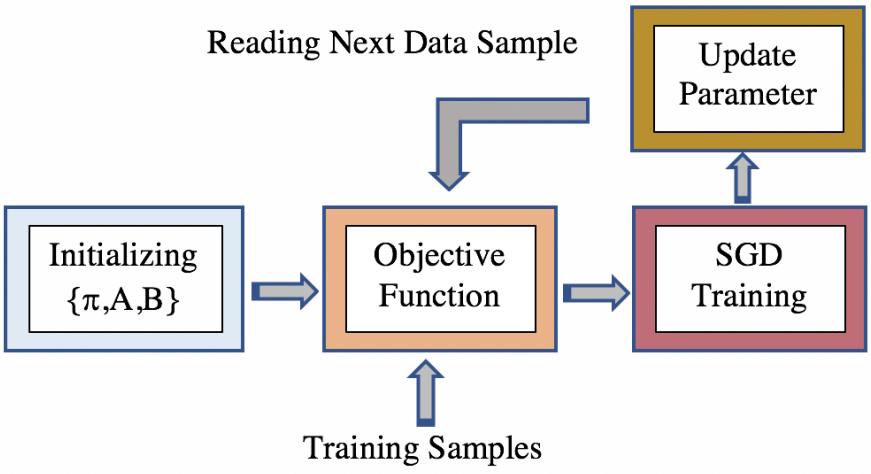


Figure 4.2: HMM-SGD Learning Process.

Figure 4.2 shows the flow diagram of the learning process. First, the model parameters $\{\Pi, A, B\}$ are randomly initialized while maintaining the probability condition, such that all numbers add up to one.

The next step is to start modeling the training data samples that involve sequences of the first 50 sessions. The learning procedure is initiated by iterating over a fixed number of iterations. Within each iteration, the objective function will be called to calculate the probability of the session actions sequence as shown in Equation 4.1. The result and the current model parameters will be fed into the gradient descent function. The gradient of the objective function with respect to the model parameters will be calculated. Later on, the parameters will be updated such that we achieve a high probability of the input data sample.

To elaborate more on the SGD training procedure, the training pseudo code is presented in algorithm 1. The main procedure begins by initializing the HMM's parameters. Then, it goes over each of the sequence data samples and calls the training procedure. Algorithm 2 illustrates the training steps that begin by calling the objective function. Then, it updates the model parameters independently by calling the gradient descent function for each of the parameters along with the objective function.

Algorithm 1 Modeling Actions Sequences

```
1:  procedure HMM_SGD(inputSequences, hiddenStates, learningRate, iterations)
2:      Create HMM Object
3:      Initialize HMM Parameters
4:          ▷ θ_old = {θ_π_old, θ_A_old, θ_B_old}
5:      trainingLength ← length(inputSequences)
6:          ▷ The first 50 sessions
7:      while iterations do
8:          for Training sequences do
9:              HMM.trainModel( sequence, θ_model, iddenStates, learningRate)
10:                 ▷ θ_model is the model current parameters
11:             iterations − −
12:         end for
13:     end while
14: end procedure
```

Algorithm 2 Training Procedure

procedure TRAINMODEL(sample, θ_{model}, hiddenStates, learningRate)

$Objective = HMM.Objective(sample)$

$\theta_{\pi\_new} \leftarrow \theta_{\pi\_old} - \mu * SGD(Objective,\ \theta_{\pi\_old})$

▷ θ_{π_old} *is the current initial probability vector*

▷ μ *is the learning rate*

▷ SGD *is a stocastic gredient descent function*

$\theta_{newA} \leftarrow \theta_{oldA} - \mu * SGD(Objective,\ \theta_{oldA})$

▷ θ_{oldA} *is the current transition probability matrix*

$\theta_{newB} \leftarrow \theta_{oldB} - \mu * SGD(Objective,\ \theta_{oldB})$

▷ θ_{oldB} *is the current emission probability matrix*

end procedure

4.2 DATASET

To test the performance of the proposed approaches, we need a data set that can be used to profile the users' behaviors based on machine log data. For that reason, we used the CERT Insider Threat Data sets [24, 29]. The CERT Division cooperated with ExactData, LLC, to create several versions of synthetic insider threat data sets. These data sets are unlabeled sets. They have both synthetic base data and synthetic malicious user data. The data sets project is sponsored by DARPA I2O [24]. The CERT data set[1] is a diverse domains data set. It is a public data set that consists of different computer-based log events data files [73]. The raw logs data sets include the following computer log events: logon/logoff, the logs of open/closed files, the logs of all surfed websites, the logs of how a user uses the thumb drive Connect/Disconnect, the logs for email messages that have been sent and received, and one file for LDAP information [73, 12, 24]. For our work, we used the r4.2 data set that has a huge variety of users event logs. Even though CERT provides r6 data sets, r4.2 has many insider users for several scenarios which is the reason that we adapt r4.2 in this work.

[1] `https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=508099`

4.3 THE PROBABILITY OF A GIVEN SAMPLE SEQUENCE

The raw log events are regenerated to be sequences of timed events, as is illustrated in section 4.5. To find the probability of each of the created sequences $Y = (y_1, y_2, ..., y_T)$, we can use one of the two Algorithms: the Naive Based Algorithm, or the Forward-Backward Algorithm.

To use the Naive Bayes method, we need to consider all possibilities of hidden states sequences and add the probabilities across all of them. Using this method is not an efficient way because it increases $O(TN^T)$ runtime. Alternatively, The Forward-Backward algorithm [72] is more efficient and it elapses $O(NT^2)$ runtime. In general, the sequence probability is a redundant process of the sum of the product of fraction numbers. The multiplication of two fractions will result in a smaller value. This fact produces small amounts that cannot be processed by computers because of the resolution capability. In many cases, it turns out to be zeros. Thus, in our work, we use the $(-log(P(Y))$ instead of $P(Y)$. The works in [73, 72] adapt these solutions for the machine resolution problem. In the training section, we will explain two more solutions that we use in our model.

4.4 MALICIOUS INSIDER'S FEATURES

To track the users' suspicious actions in any organization, we need to choose the right features that help in profiling and modeling a users' behavior pattern. According to the work of [73], it is more accurate to adopt several different domains of actions events' logs to model users' behaviors. This would combine many trends of users' attitudes. For this reason, it is beneficial to take into consideration the computer-based users' actions events. For example, we can model different kinds of event logs to profile users' behaviors. The logs include when a user often logs on and logs off, what sites are used most frequently by a user, who the possible receivers of the user's email messages are, and more, allowing us to profile a user's behavior. Discovering an unusual pattern in the activities events log leads

to discovering possible insiders threat attacks. The more frequently these patterns occur, the more accepted they are [66].

The main purpose that we aim to achieve is to reform the computer-based event log features, from CERT division data sets, which are used with the proposed models. We use two kinds of machine learning models the hidden Markov model and the HMM-SGD. Both of the adopted models work with a sequence-shape data sample. Therefore, we preprocess the multi-domains log events and produce sequences that will be fed to the HMM models. In this paper, we adopt all features that are included with the CERT data set to create session-based, day-based, and week-based data samples of users' action events.

The next section will illustrate the extraction of features and the implementation of the proposed approaches as well.

4.5 Preprocessing of Log Data

At the beginning of this section, we will explain how we preprocess the raw events' log data from CERT datasets. The preprocessing procedure starts from reading the log files from different log domains of each user and ends with the generation of new encoded action event sequences that present user behaviors. The preprocessing phase has four essential stages: Filtration, Encoding, Merging and Extracting. Figure 4.3 shows the general overview of our preprocessing stages. Each step of the preprocessing is designed to be an independent module. Thus, it can be updated without affecting the other stages.

4.5.1 Features Selection and Extracting

The CERT dataset provides comma separated value (CSV) log files of five different domains [24]. We selected all of the features provided by files. Each one of these files provides log activities of users from a specific domain. For instance, the Device file has the log events that indicate when and where a user has connected or disconnected a removable drive to or from a machine. The adapted CERT features include: Logon (User logged onto

a computer/unlocked a computer); Logoff (User logged off a computer); File (User copied a file to a removable drive); Email (User sent an email); Website (User visited a website); Connect (User inserted a removable drive); Disconnect (User removed a removable drive).

Figure 4.3 illustrates the main stages of selecting and extracting features. There are four stages (Filtration, Encoding, Merging, and Extracting) that are used to process the raw features. The CERT datasets are big datasets that require a large memory space. For example, a machine with 16 GB RAM cannot hold some of the preprocessing steps especially during the Filtration stage. To overcome this issue, the data filtration performed during loading the CSVs log files from the hard disk.

Two working environments are used in this paper:

1. "R" is used for pre-processing CERT dataset and generating the sequence data samples.

2. The generated data samples are modeled using the Python environment.

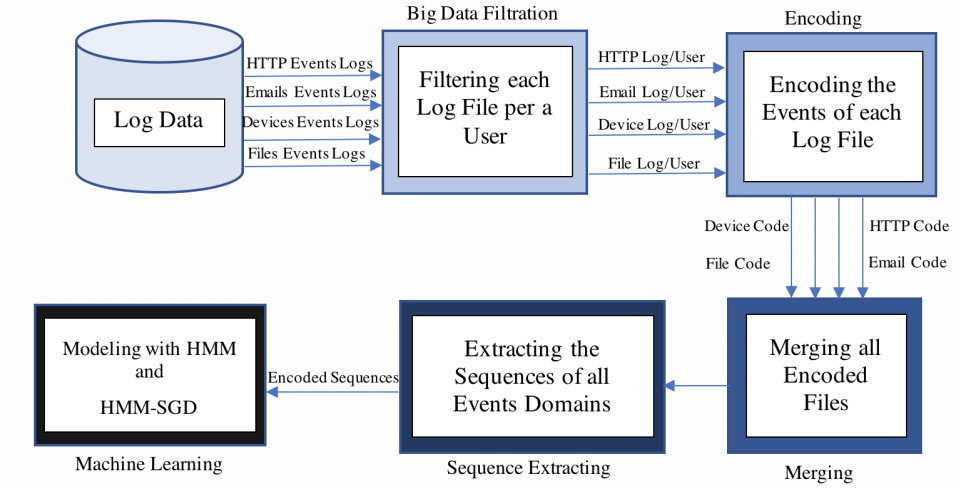


Figure 4.3: The general platform of the insider threat detection system.

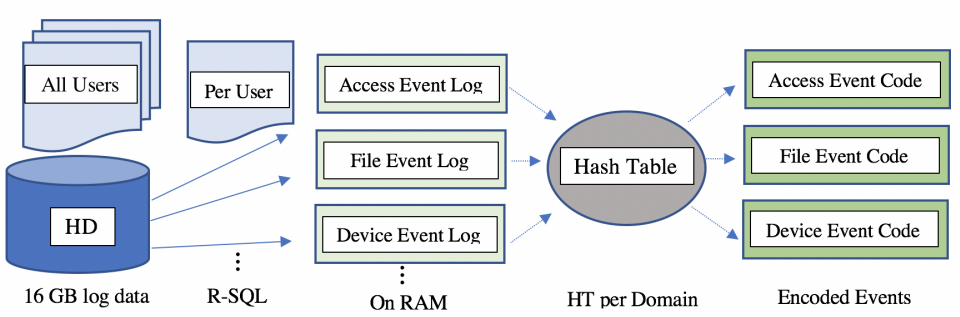


Figure 4.4: Filtering and encoding multi domains logs data.

The Filtering, Encoding and Merging processes are described as follows:

1. The system starts with filtering the log files of a given user. It uses “R-SQL” to filter the data while loading these files from a hard disk. Using this technique gives the ability to use the available memory size without any issues as shown in Figure 4.4. The developed preprocessing approach compensates the memory limitations by systems hard disk.

2. The filtered events are encoded sequentially with a hash Table . For instance, if the user “AAM0658” logs in to the system and performs several activities on his machine. The log data of these actions will be encoded as a sequence of numbers. Each number stands for a specific action made by the user.

   In HMM context, each code refers to an index in the symbols matrix of Hidden Markov Model. For instance, a representation of one of the encoded session-based symbols of user “AAM0658” are illustrated in Figure 4.5.

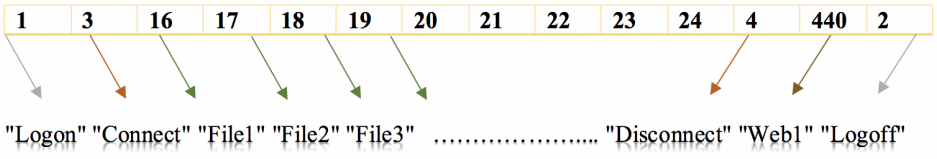


Figure 4.5: A session sequence of “AAM0658”

3. The encoded events of different domains are merged based on their time-stamp to generate a big vector of symbols.

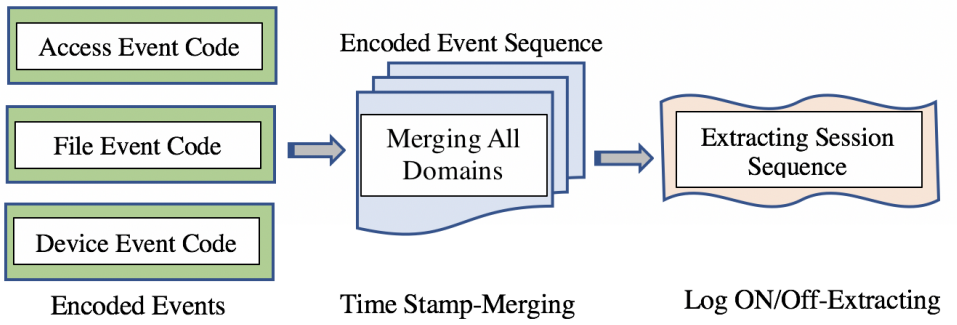


Figure 4.6: Merging encoded events followed by extracting session-based sequences.

4. The last step is extracting the data samples. The big symbols vector is evaluated to generate three different samples: session-based, day-based and week-based samples. The session-based are determined by tracking the (log on/ log off) events codes. All symbols between log on/log off codes are considered as a session-based sequence.

   The day-based and the week-based samples are obtained by aggregating action events per a day or per a week.

   Hour, day, week, year and several others features are created to facilitate the pre-processing operations to generate the session-based, day-based , and the week-based sequences.

   Finally, the resulted sequences are saved in a data frame to be modeled later with HMM and HMM-SGD.

4.6 Model Structures and Results

The work presented here utilizes two models: the HMM and the HMM-SGD models. In this section, only the results of the HMM-SGD model is presented because the difference

between the performance of the two models is minimal. However, the HMM-SGD model has more tuning flexibility due to the presence of more hyper-parameters.

The structures of HMM and HMM-SGD are also described.

4.6.1 HMM STRUCTUE

HMM experiments were conducted with three hyper-parameters: the number of hidden states, the maximum number of iterations and the number of training samples. The list below shows the combinations of the used hyper-parameters along with the structure of HMMs.

1. HMMs are implemented with $10, 20, 40, 50, 60$ hidden stats.
2. Three detection systems are implemented: the session-based, the day-based and the week-based. Each one of these models is trained using the Baum-Welch algorithm for 20 iterations.
3. We fix the training sets as follows:
   - Session-based system: the first 50 sessions.
   - Day-based: the first 35 day samples.
   - Week-based: the first 5 weeks samples.
4. After the training process, we find the probability of each sequence P(sequence) using the forward algorithm.
5. To avoid the resolution machine problem, we use $(-log(P(sequence)))$ instead of $P(sequence)$.

4.6.2 HMM-SGDs' Structure and Results

The HMM-SGD models are trained with four hyper-parameters: the number of hidden states, the maximum number of iterations, the number of training samples, and the value of the learning rate.

The structure of the HHM-SGD models and the hyper-parameter combinations are similar to HMM modes. The only difference is the learning rate. The HMM-SGD models are trained with a 0.01 learning rate.

Similar to the baseline HMM, the probability of each sequence P(sequence) is calculated using the forward algorithm. The results are evaluated using the ROC curve to see the overall performance of the proposed detection approaches.

4.6.3 Session-based Model Results

The session-based sequence is a low level granularity sample. The representation of user actions per session are too narrow to consider many of the users' activities, compared to a day- or week-based sequence. Moreover, the insiders usually distribute their actions over several sessions so that no one can recognize their anomalous behaviors. However, the session-based system provides the shortest detection time.

Although the session-based results look weak, the model shows a very good performance by evaluating the sessions that are near to or surround the labeled sessions with very low probability values, (for more details see study case in section 4.8). This has inspired us to come up with the idea of a sliding window technique to optimize the model performance.

4.6.4 A Sliding Window Technique

To optimize the models evaluation process, we propose a novel Sliding Window Technique. This approach provides the flexibility to monitor the recent history of user behaviors. For example, to evaluate the session sequence of user "AAM0658" (Figure 4.6), the sliding window will be used to see the history of the current sequence based on a window size.

Thus, instead of considering just the predicted probability of a sequence, a sliding window gives a broad vision of user behaviors.

The proposed technique can also be used to see the future changes of behaviors regarding a current session. For instance, if we want to evaluate session 100 of user "AAM0658", we can see the changes in his behaviors between sessions 100 and 110, using a window of size 10.

In this work, we use the Sliding Window to monitor the recent history of user behaviors.

4.6.5 Day-based Model Results

The day-based detection system shows a better performance compared to a session-based system. The better performance comes from the fact that a day-based sample has more action events than a session-based sample. Monitoring more events enables the model to evaluate more users' activities which usually come from multiple domains. For example, the model evaluates more emails, files, and even more web sites which improve the model to perform well.

4.6.6 Week-based Model Results

The week-based detection system shows the best results compared to a session-based system or day-based system. The improvement of the model performance comes from the fact that a week-based sample has more events to monitor than what events are in a session-based or day-based sample. A week-based sample reflects a broad distribution of user behaviors and that helps the model to show good performance.

4.7 Model Evaluation

The CERT data set has several scenarios and provides description files for insider's events. These files specify the events that are considered as malicious behaviors. That data is used as truth labels to evaluate the work.

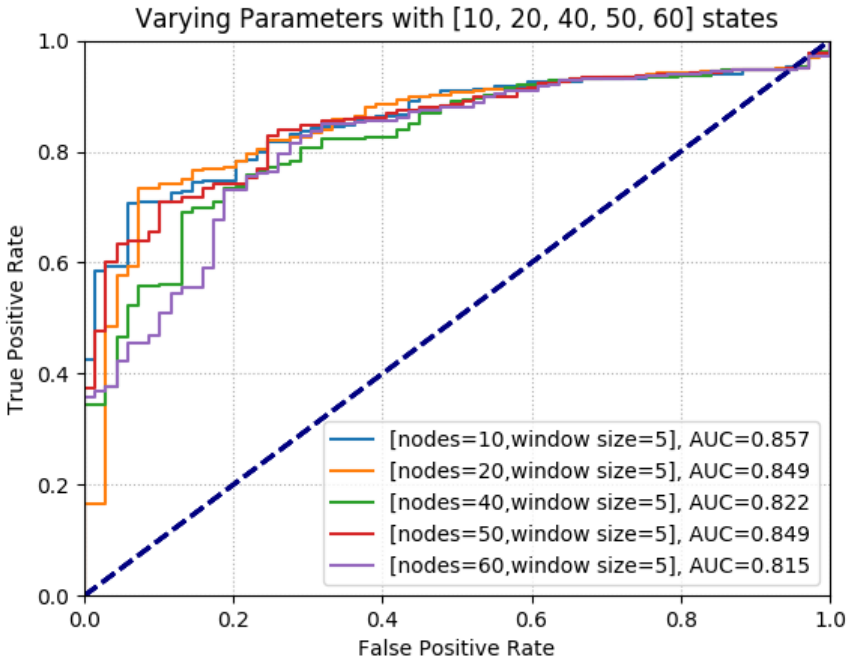


Figure 4.7: 5 Samples Window Size.

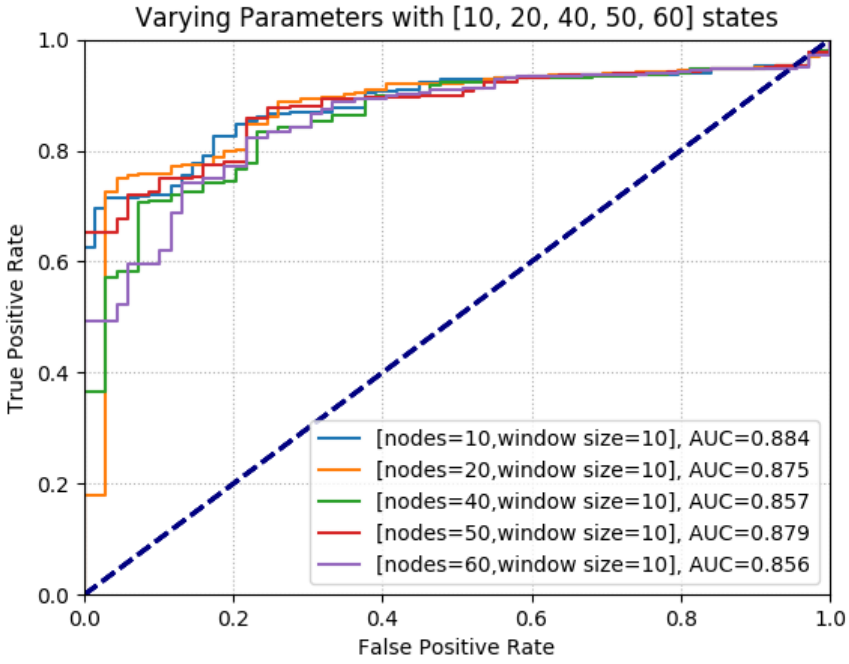


Figure 4.8: 10 Samples Window Size.

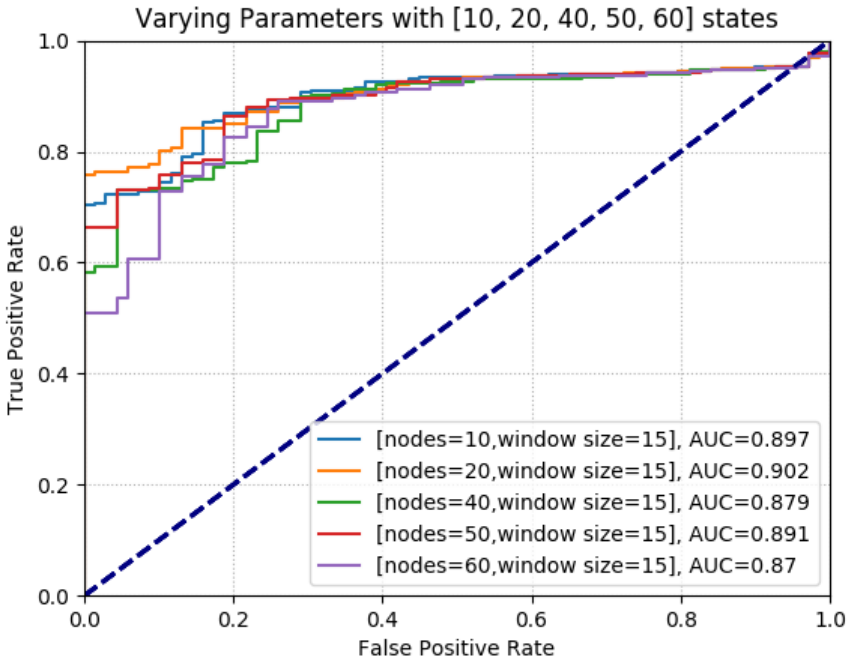


Figure 4.9: 15 Samples Window Size.

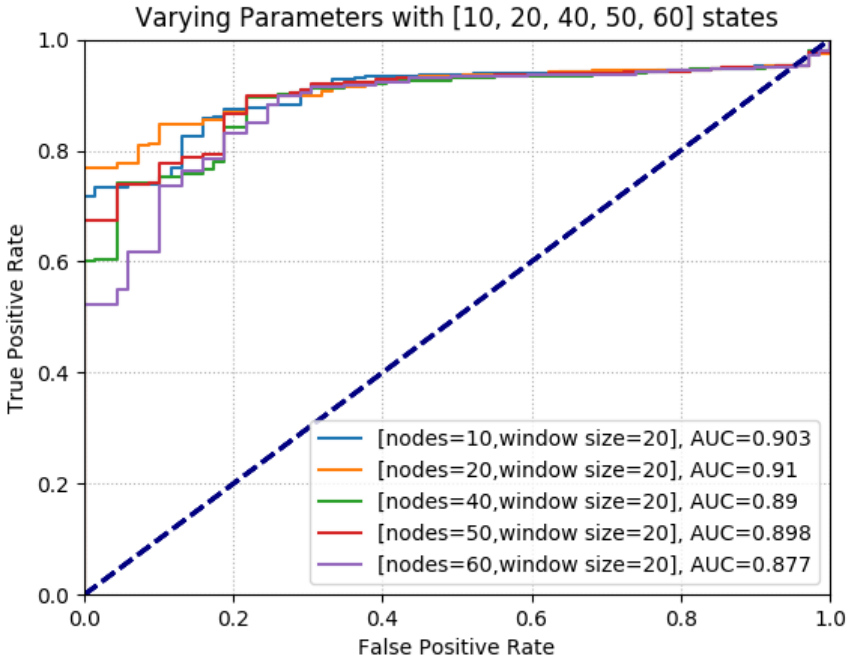


Figure 4.10: 20 Samples Window Size.

The truth labels of thirty users are used to evaluate the presented work. Those users are insiders according to the definition of scenario one. The insiders attack in scenario one occurs as follows: a User begins to log on after office hours, starts using a removable drive and then begins uploading data to wikileaks.org.

The generated session-based, day-based, and week-based data sets are labeled manually and using a labeling system according to the truth label data.[2]

4.7.1 Normalization

The raw event logs of each user are separately preprocessed as described in section 4.5.1. Thus, the baselines of normal behaviors are different among users. To evaluate our work, we normalize the predicted probabilities of a user's actions according to the training data samples. The structure of the training setup is illustrated in section 4.6.1.

For every user, we normalize the predicted probabilities of testing data points as follows:

1. Run the model with the training data sets: session, day, and week-based granularity.
2. Averaging the predicted probabilities to calculate the baseline of normal behaviors.
3. Using the normal baseline to normalize the predicted probabilities of testing data samples.

Then, all normal baselines are normalized to be in the same scale.

4.7.2 Evaluation with Roc

The ROC procedure takes all truth labels of user session sequences along with the *P(session sequence)* of each sequence and applies a variety of threshold values to draw two dimensions of the ROC curve. Each point of this curve represents a result of the threshold-based

[2]Note: Every procedure in this work is built from scratch. No code is provided from any other work.

comparison: how many data samples have been evaluated as normal or abnormal compared to the truth labels. The area under the curve (AUC) is also used to assess the model performance. AUC is a calculation of the area under the plotted ROC curve. The more the AUC is close to one, the more the model has a good performance and vice versa.

Figures 4.7, 4.8, 4.9, 4.10 show the ROC curves of applying sliding window with session-based data samples. They present the use of different window size. The experiment is implemented with five different model structures and four different window sizes. These include 10, 20, 40, 50, and 60 hidden states and 5, 10, 15, and 20 window sizes. Also, the AUC is presented under each model structure with a different color.

4.8 CASE STUDY

Table 4.1: User "MCF0600" Data Samples Statistics

Granularity-based	**Session**	**Day**	**Week**
Data Samples	308	246	41
Malicious Samples	3	3	1
Training Samples	50	35	5
Malicious Labels	276, 278, 280	221, 223, 224	38

To investigate more about the detection function of the model, a case study is illustrated with more details. The User "MCF0600" is selected as a case study which is the same case study as in [73]. As mentioned in section 4.7, the attack is started when user "MCF0600" begins to log in after office hours, starts using a removable drive, and then begins uploading data to wikileaks.org. User "MCF0600" is considered one of the insiders according to scenario one. The user has malicious behavioral truth labels, so we can use his log data to evaluate the models.

Table 4.1 previews information about the data sample representations for user "MCF0 600": session, day and week-based data samples. The information includes the total num-

ber of samples, the number of malicious samples, the number of training samples, and labeled indexes of each data set, section 4.5.

Figure 4.11 (a), (c), and (e) shows the predicted -Log(Probability) of the three detection systems. The normal samples are colored blue while the malicious samples are colored red. The results are for the testing data sets, So the labeled indexes are different compared to the numbers in Table 4.1.

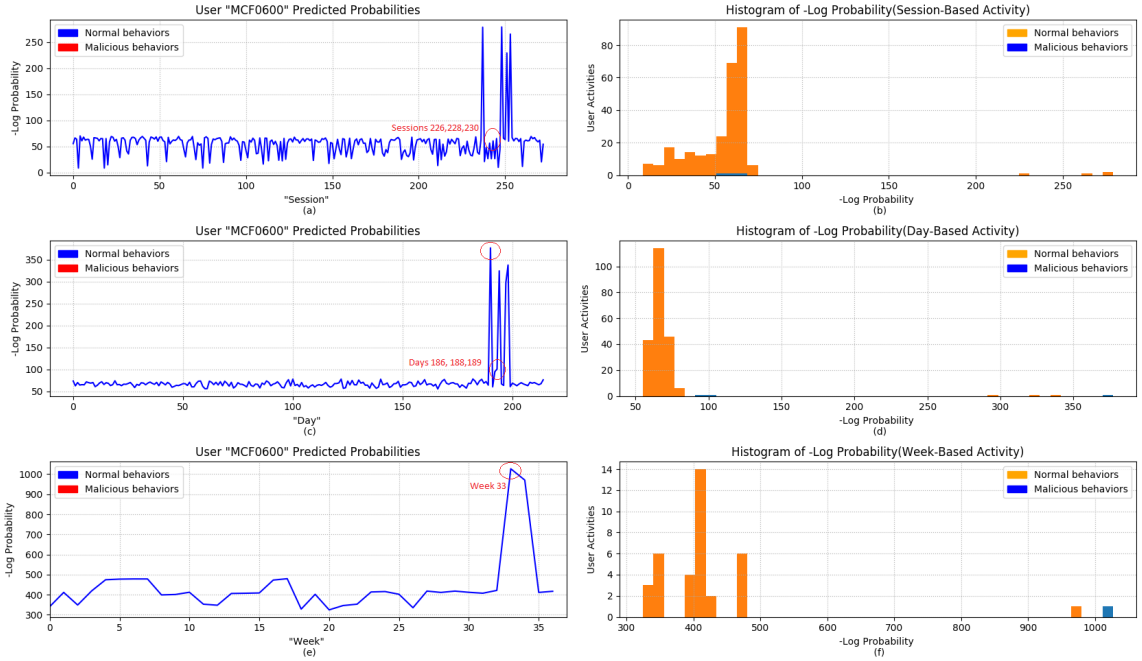


Figure 4.11: Histogram and scatter plots of probability scores of "MCF0600." Session, day, and week-based data samples are used to evaluate the users' behaviors.

Figure 4.11 (a), (c), and (e) also presents the histogram distributions of the predicted probabilities with a class color, blue for abnormal behaviors and orange for normal behaviors. The session-based system evaluates the labeled sessions with a high probability[3], sessions 226, 228, and 230. On the other hand, the model evaluates the unusual copying of files or surfing unusual web sites with very low probabilities as shown in Figure 4.11 (a), sessions 221, 232, 235, and 237.

[3]High probability means low -Log(Probability).

To improve the model performance, a sliding window technique is used with the predicted probabilities to analyze the effect of the history behavior on the current behavior.

The ROC curve is used with and without a sliding window. The first score is 0.2 while a full score, 1, is the result of using a sliding window. A Window size of 5, 10, 15, and 20 is used and the result is a full score, 1.

Using the sliding window with the session-based system gives a fast detection system with very good accuracy.

CHAPTER 5

INSIDER THREATS DETECTION USING CNN-LSTM MODEL

Malicious insider activities threaten various government agencies and private organizations. This chapter presents a novel approach to detect malicious behaviors. We propose the use of a granularity level to represent users' log data: textual session-based data samples. The user's behaviors are modeled using character embeddings and a deep learning model that consists of CNN and LSTM. Character embeddings are used to represent the input samples. Then, a convolution layer is used to capture local tri-gram features from the input samples, followed by an LSTM layer to consider the order of these given features (tri-grams). We conduct experiments using several variations of model architectures with no handcrafted features. The proposed model is evaluated with a subset of CERT Insider Threat dataset, r4.2. The results show that performance improved with high AUC values.

5.1 INTRODUCTION

In this chapter, the raw data from five different domains, "Logon/ Logoff," "Connect/ Disconnect," "website links," "Emails," and "local files," are pre-processed to generate new data samples: textual session-based sequences. The generated data samples are level granularity samples. They have short length activity sequences compared to day-based or week-based samples. The new representation indicates the detection time unit. For instance, a session-based sample means the user activities will be evaluated after each session. The presented data samples are different from the work in Chapter 4. The data is presented as

session-based encoded samples in Chapter 4, while in this work the textual session-based data samples are proposed. The new representation gives the ability to go over the real textual meaning of the event logs using **Natural Language Processing NLP**; this ability would improve the model performance to detect the malicious activities.

Natural language processing (NLP) enables computers to perform a wide range of natural language-related tasks, such as classification. The traditional methods, which have been utilized to solve these NLP problems, such as SVM and logistic regression, were trained on very high dimensional data, or data with many features. These traditional machine learning based NLP systems relied heavily on hand-crafted features which in turn are time-consuming and often incomplete.

In the last few years, neural networks based on dense vector representations have been producing superior results on various NLP tasks including classification [4]. An advantage of using neural networks is that they require no hand-crafted features and enable automatic feature representation learning.

In this chapter, we proposed to use a combination of Convolutional Neural Network (CNN) [50] and Long Short-Term Memory (LSTM) [38] to model user actions and detect the abnormal behaviors. A user's behavior is represented as a series of activities performed within the organization environment.

The goal of this work is to develop an insider detection system that has the ability to detect malicious activity in a relatively short time, which is not addressed by other works. Our contributions can be summarized as:

1. Processing the raw log data to be in textual session-based sequences.

2. Proposing the neural network model to process the sequence data samples and detect the malicious activity.

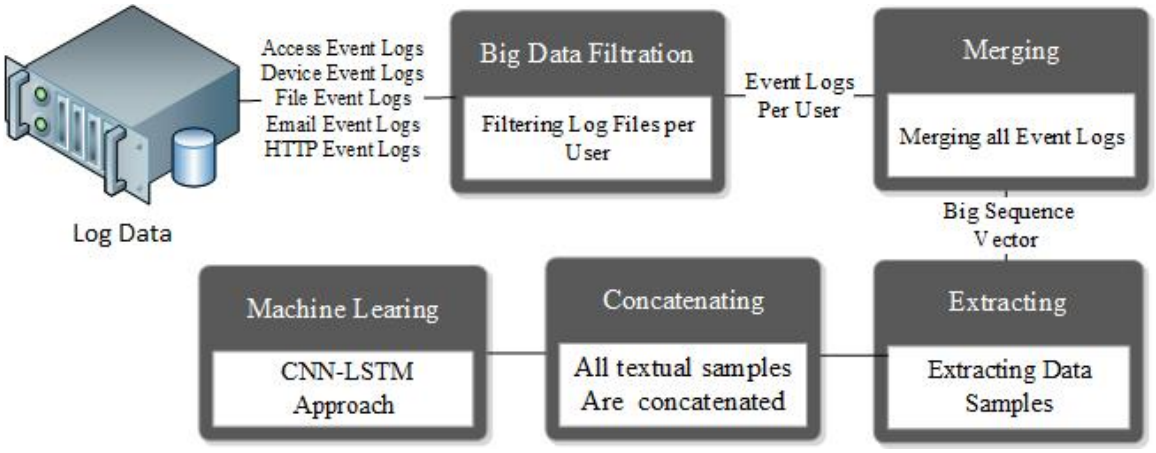


Figure 5.1: The general platform of the insider threat detection system.

5.2 Preprocessing of Log Data

The preprocessing phase has four essential stages: Filtration, Merging, Extracting, and Concatenating. Figure. 5.1 shows the general overview of our preprocessing stages. We design each step of the preprocessing phase to be an independent module. Therefore, it can be updated without affecting other stages.

5.2.1 Features Selection and Extracting

The adapted CERT features include: Logon (User logged onto a computer/unlocked a computer); Logoff (User logged off a computer); File (User copied a file to a removable drive); Email (User sent an email); Website (User visited a website); Connect (User inserted a removable drive); Disconnect (User removed a removable drive).

The CERT datasets are big datasets that need a large size memory; a machine with 16 GB RAM cannot hold some of the preprocessing steps, especially during the Filtration stage. Therefore, to overcome this issue, the data filtration is performed during the loading of data files. The Filtering and Merging processes are described as follows:

1. The system starts with filtering the log files of a given user. The Filtration process uses “R-SQL” technique to filter the data while loading these files from a hard disk.

Using this approach gives the ability to use the available memory size without any issues as shown in Figure. 5.2.

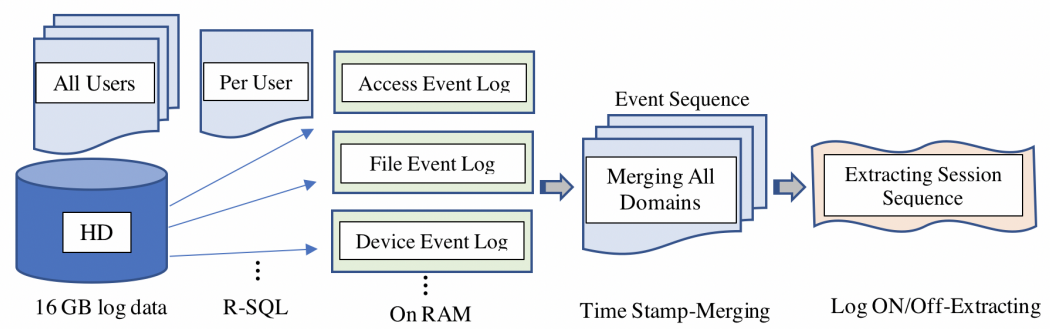


Figure 5.2: Preprocessing of Event Logs.

2. The filtered events of different domains are merged based on their time-stamp to generate a large vector of textual action events.

3. The large event logs vector is processed to generate the new session-based sequences by tracking the (log on/ log off) events. All logs between log on/log off events are considered as one session-based sequence.

4. Then all users' data is merged into one dataset. Each session is a long string of all user's activities such as: web site address, file name, the "To" section of an email, etc. In total we have 30 users' profiles; all these users have 7587 sessions. Only 69 data points (sessions) are labeled as an insider; the rest (7518) data points are examples of normal behaviors.

5. In order to train a neural model for the data, we need to transform the dataset from its textual shape into a numerical form. We do that by transforming each character into a character embedding. Character embedding (or word embedding) is a class of approaches for representing words or documents using a dense vector representation that captures something about their meaning. The main idea behind this approach is that each word or character can be represented by the meaning of its neighbors [25].

There is a linguistic theory behind the approach, namely the "distributional hypothesis" by [36]. Compared with traditional word representations, word embedding is an improvement over simpler bag-of-word (BOW) models. BOW representations result in large and sparse vectors (mostly 0 values) that describe documents but not the meaning of the words.

5.2.2 Over-sampling of Minority Group

The CERT dataset is imbalanced data. The classes of the data points are not reasonably represented. For instance, the malicious data samples are too few compared to normal examples. Generally, real-world data sets consist of a high ratio of normal examples and a small ratio of abnormal examples [16].

After splitting the dataset into training and testing sets, we end up with fewer examples of insider data points. This is a textbook example of an unbalanced dataset, which will affect the model performance. To solve this issue, oversampling the minority class observations may improve the quality of the model. By oversampling, models are sometimes better able to learn patterns that differentiate classes. We up-sample the insider labels using the SMOTE algorithm (Synthetic Minority Oversampling Technique) [16]. At a high level, SMOTE creates synthetic observations of the minority class by:

- Finding the k-nearest-neighbors for minority class observations (finding similar observations)
- Randomly choosing one of the k-nearest-neighbors and using it to create similar, but randomly tweaked, new observations.

When up-sampling using SMOTE, we do not create duplicate observations. However, because the SMOTE algorithm uses the nearest neighbors of observations to create synthetic data, it still bleeds information. If the nearest neighbors of minority class observations in the training set end up in the validation set, their information is partially captured by the

synthetic data in the training set. As a result, we over-sample only the training dataset. By oversampling only the training data, none of the information in the validation data is used to create synthetic observations. So, these results should be generalizable.

5.3 Natural Language Processing with Neural Networks

NLP is the bridge between human language and machines. It is a combination of machine learning and linguistics to understand written or spoken languages. NLP is a way to represent, analyze, and interpret the textual data, such as letters, words, and sentences.

In the last few years, neural networks based on dense vector representations have been producing superior results on various NLP tasks including classification [19]. An advantage of using neural networks is that they require no hand-crafted features and enable automatic feature representation learning.

In this chapter, we proposed to use a combination of a Convolutional Neural Network (CNN) [50] and Long Short-Term Memory (LSTM) [38] to model user actions and detect abnormal behaviors.

5.3.1 Convolutional Neural Network

Convolutional Neural Networks (CNN) have recently been shown to achieve impressive results on the practically important task of sentence classification [30]. For most NLP tasks, CNN plays the role of feature extractor by extracting higher-level features from constituting words or n-grams to create a useful latent semantic representation of the sentence [19]. Initially, CNN was designed to be used in image processing tasks. Therefore, the input is expected to be a "2-D matrix" representing image pixels. For NLP tasks, instead of image pixels, the input is sentences or documents as sequences of tokens. In order to apply CNN, the input needs to be represented as a matrix where each row of the matrix corresponds to one token, usually a word or a character. Thus, each row is a vector that represents a word or character. Typically, these vectors are embeddings.

Instead of hand engineering our features to classify whether a session is a normal or abnormal behavior, we use a CNN as the feature extractor. Figure 6.26a illustrates our CNN model architecture. The model is comprised of a filters layer and a pooling layer. The filters layer consists of 32 filters of size 3, and their width would be equal to the embedding dimensions (100 dimensions) used to represent each character in the session sequence.

This layer performs convolutions on the input sentence matrix and generates 32 feature maps. The largest number from every two neighboring cells in the feature map is selected using a 1D-max pooling to produce the compressed feature map. These 32 compressed feature maps are concatenated to form one feature matrix. This feature matrix would represent the higher-level features of the input session sequence and would be passed into the LSTM layer.

5.3.2 Long Short-Term Memory network

A Long Short-Term Memory network (LSTM) is a particular type of Recurrent Neural Network (RNN). A recurrent neural network is a neural network that attempts to model time or any other sequence, such as language. One problem exists with standard RNN as the distance between words or sequences values increase, i.e., they are separated by a large number of other words or values. Modeling such dependencies will lead to the vanishing gradient problem (or exploding gradient problem). LSTM is capable of overcoming such shortcomings of standard RNN by learning long-term dependencies using a new structure called a memory cell. A memory cell is composed of three main gates: an input gate, a forget gate and an output gate. The weights of these gates will model the interactions between the memory cell itself and its environment. As our CNN layer learns and outputs the most important features of the session sequence, the LSTM considers the order of the user's actions in a specific session.

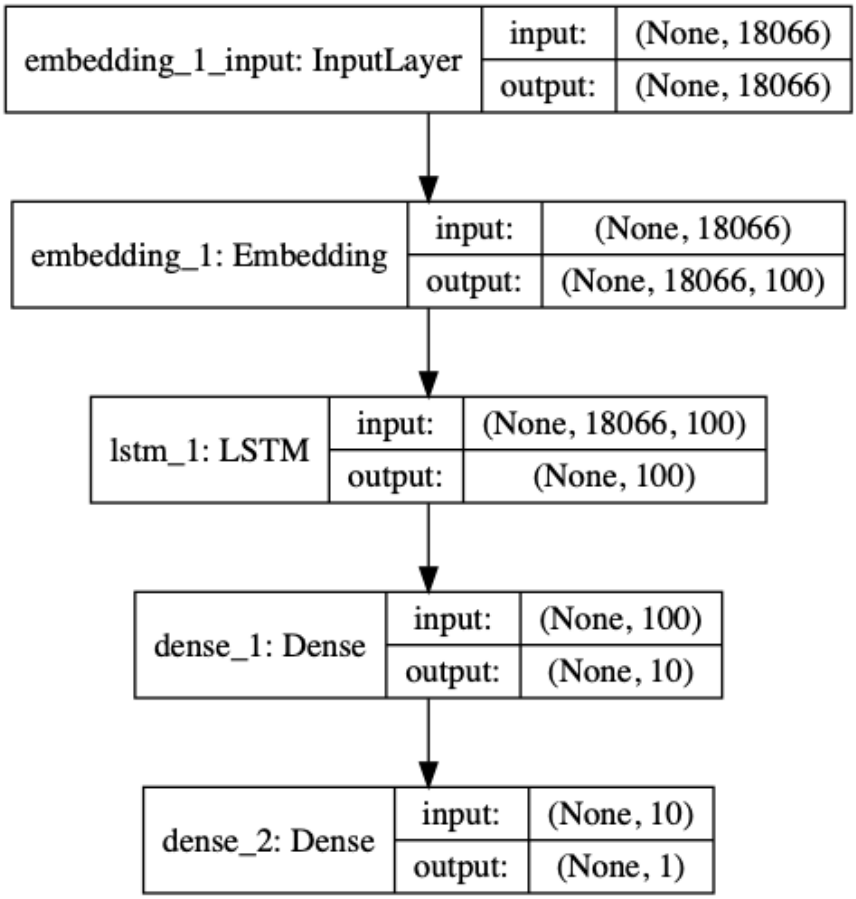


Figure 5.3: The architecture of LSTM model with input/output dimensions.

5.4 Model's Architecture

Throughout our work, we tried three model architectures:

5.4.1 LSTM Model

The first model is a one-layer LSTM followed by a densely-connected NN layer with a "sigmoid" function. In this model, we utilized char embeddings to represent the input session sequence. We tried to let the model learn these embeddings by considering them as another hyper-parameter that the model needs to learn. These are the hyper-parameters of this model: 100 LSTM units, batch size: 32 and epochs: 10. Figure 5.3 shows the proposed LSTM architecture with the input/output dimensions.

5.4.2 CNN MODEL

The second model is a one-layer CNN followed by a densely connected NN layer with a "sigmoid" function. In this model, we utilized char embeddings to represent the input session sequence. We tried to let the model learn these embeddings by considering them as another hyper-parameter that the model needs to learn. These are the hyper-parameters of this model: CNN filters: 32, filter size: 3, pool size: 2, batch size: 32 and epochs: 10. Figure 5.5 shows the input/output dimensions of CNN architecture.

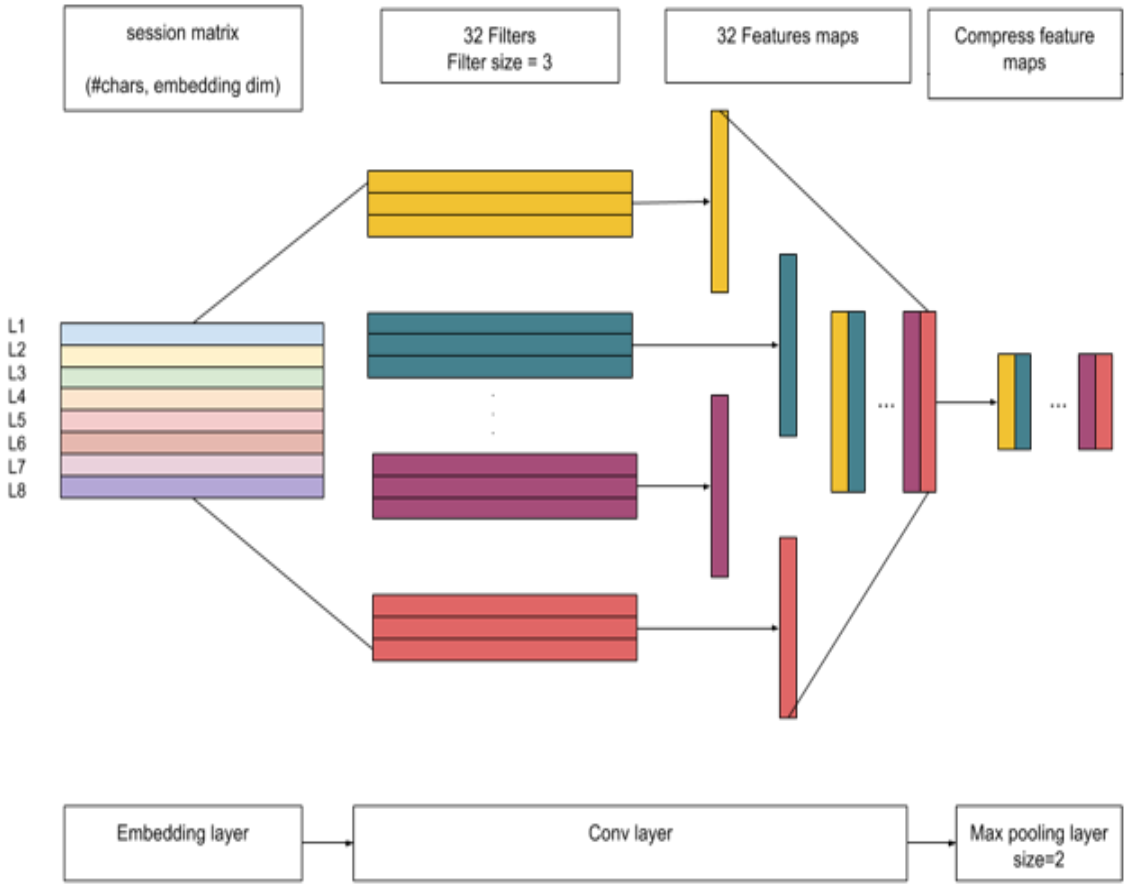


Figure 5.4: CNN Feature Extractor Architecture.

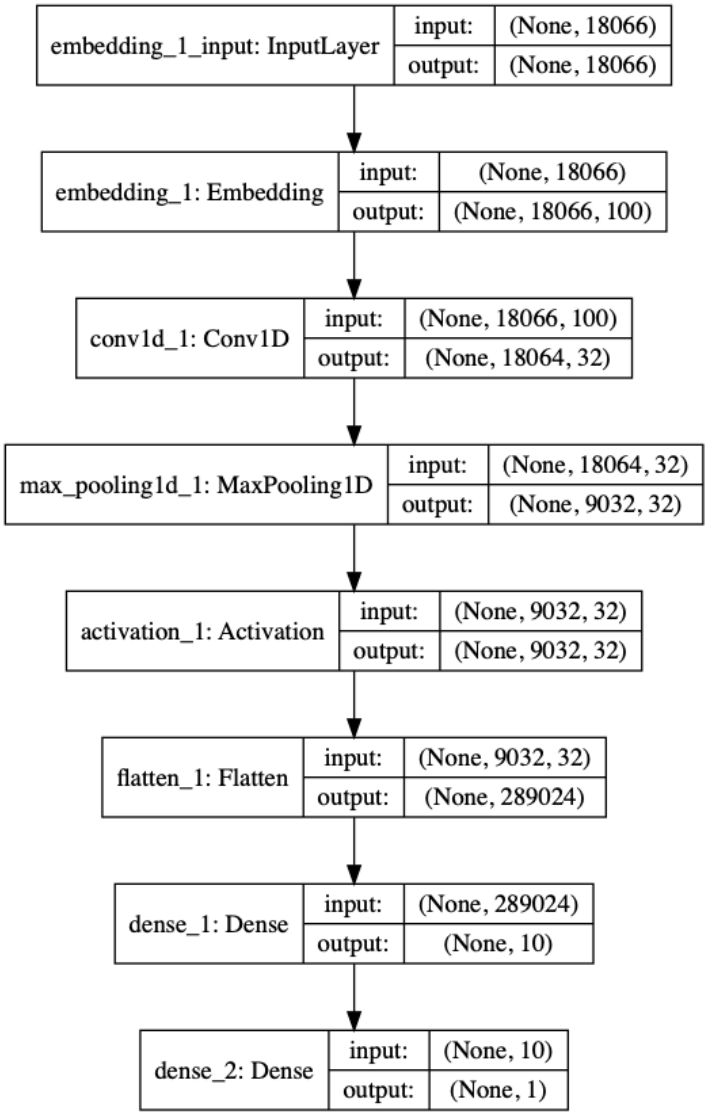


Figure 5.5: The architecture of CNN model with input/output dimensions.

5.4.3 CNN-LSTM Model

The third architecture is a combination of the first and the second architectures as shown in Figure 5.6 and Figure 5.7. There are four layers in the proposed architecture:

1. The first layer is the input layer, where the input textual session-based sequence is converted from a list of tokens (i.e., char) into a list of the char index. The char index is just the location number of that char in the dictionary of the unique characters that occur in the entire sessions.

2. To solve the problem of invariant session lengths, each input session is padded with zeros to make all session lengths equal to the longest session in the dataset. The output of this layer would be a vector with the length equal to the longest session. The keras library [19] is used to obtain the vocabulary of character indexes and to pad the input session to get the same fixed length. The output of this layer is a 2d matrix. Each row is a 100-D char vector representing each char in the input session.

3. The third layer is a conv1D, convolutional layer. This layer applies 32 convolutional filters (filter size = 3 and activation = "relu") to the embedding matrix input. Each filter outputs a feature map vector. The length of these feature map vectors is equal to the length of the input session (i.e., the number of characters). All these 32 feature map vectors are concatenated to form a feature map matrix, and the dimensions of this matrix are equal to the number of characters in the input session by the number of feature maps. We are using a Maxpooling layer with size = 2 to compress the feature matrix.

4. The fourth layer is an LSTM with a hidden state of 100 units. In this layer, we reused the same LSTM for each row in the feature matrix. The LSTM cell maintains a hidden state and a cell state (i.e., memory cell) within it that passes forward to the next step. However, there is only 1 set of parameters being learned. Those parameters need to be able to handle all steps, conditional on the current input, hidden state, and cell state. The cell state is not an output; however, it is passed forward as an input to the next step. The hidden state is passed to the output as well as to the next step.

5. To make the prediction, we use a regular densely connected NN layer with a "sigmoid" function to squeeze the output feature vector from the LSTM.

The hyper-parameters of the third architecture are set as follows: CNN filters numbers: 32, filter size: 3, pool size: 2 and activation function is "relu". The model is trained using "Adam" optimizer for ten epochs with a batch size of 32.

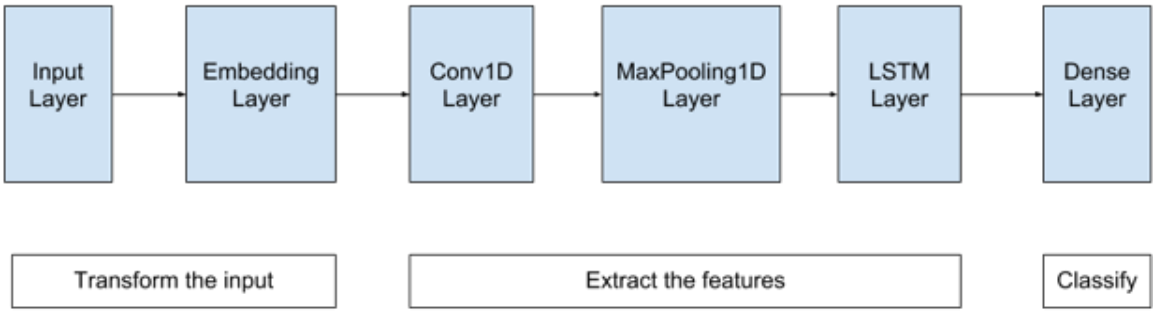


Figure 5.6: The architecture of CNN-LSTM model.

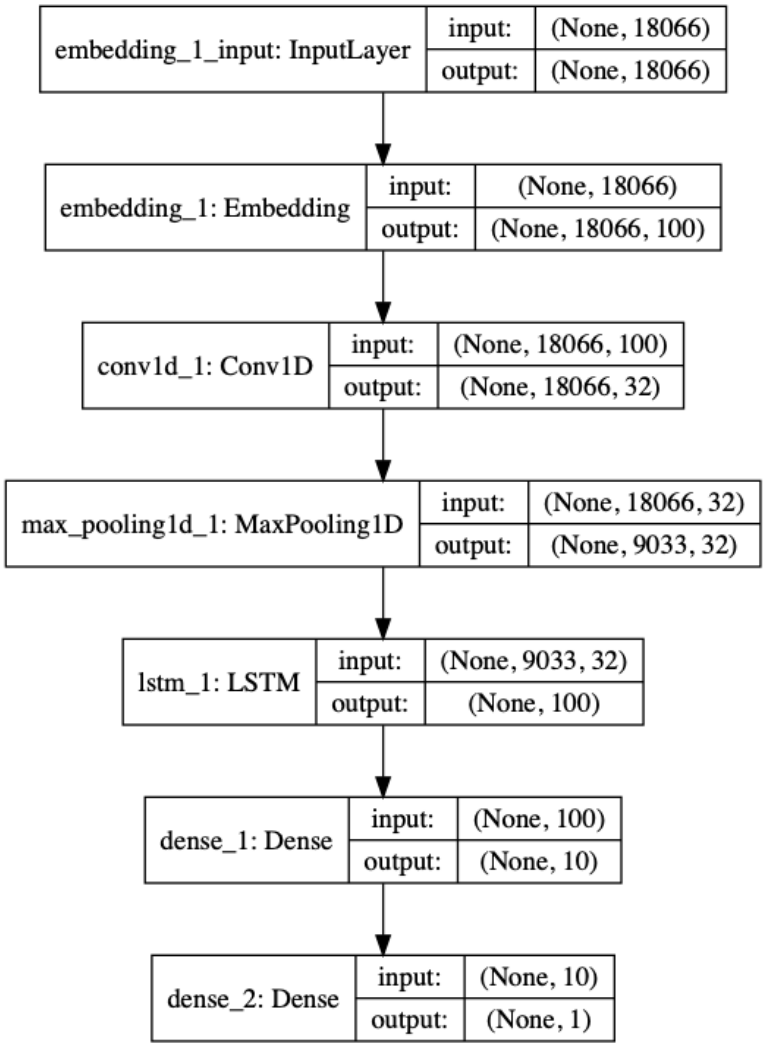


Figure 5.7: The architecture of CNN-LSTM model with input/output dimensions.

5.5 System Evaluation: Cross-validation Approach

We train and evaluate the work, CNN, LSTM, and CNN-LSTM models, using 10-fold subject-based cross-validation method. The advantage of this approach over a traditional one, where dataset is splitted into just two folds, is that all samples are used for both training and testing, and each fold is used for validation and testing exactly once.

We set up our 10-fold cross-validation experiment as follows:

1. Users' datasets are divided into 10 groups. Every group has totally separated subjects different from others.

2. We train and test the models by iterating on the ten 3-subject datasets. The proposed procedure concatenates every eighth 3-subject datasets and use them as a training dataset. The remaining two datasets are used for validation and testing.

5.6 Results with Cross-validation Approach

This section presents the 10-fold cross-validation (CV) approach. The CV approach examines the models using unseen subject-based datasets, which are not considered in the training phase.

In the conducted experiments, we consider thirty users. The data of these users are divided into 10 groups. Each group has the log data of three users. The cross-validation approach is performed in 10 rounds. Within each round, we consider the dataset of group i as test dataset, and the dataset of group $i + 1$ as validation dataset. Then, we concatenate the rest of the 8 datasets (eight groups) into one training dataset. For example, in round one, the dataset 1 is used as a testing dataset. The dataset 2 is used as a validation dataset. Then, the remaining eight datasets are concatenated to form training datasets 1. In round 2, the dataset 2 is used as a testing dataset. The dataset 3 is used as a validation dataset and the remaining datasets are concatenated to form a training dataset 2. The same procedure is performed until the 10th round where dataset 10 is used as a testing dataset. The dataset 1 is

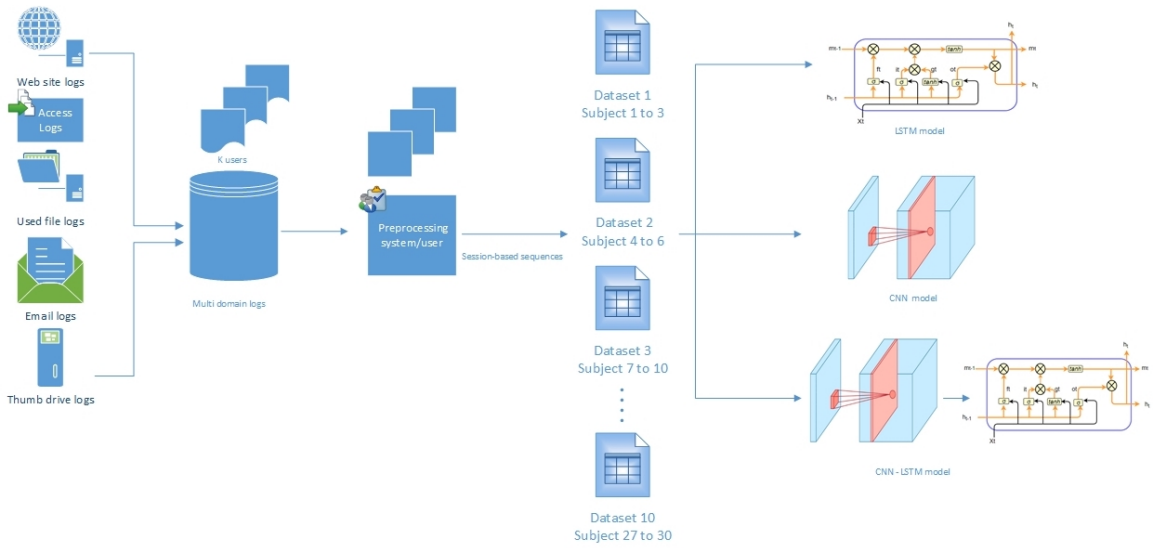


Figure 5.8: 10-folds Cross validation approach. Profile session-based datasets of 30 users are grouped into 10 parts. Every part has dataset of three subjects. The cross-validation approach are performed in 10 rounds. For example, in round one, the dataset 1 is used as testing dataset. The dataset 2 is used as validation dataset.The remaining eight datasets are concatenated to form training datasets 1. The same procedure is performed until the 10th round where dataset 10 is used as testing dataset. The dataset 1 is used as validation dataset. and the remaining nine datasets are concatenated to form the 10th training dataset.

used as a validation dataset and the remaining eight datasets (2-9 groups) are concatenated and used as a training dataset.

Each of the proposed models is trained with 10 epochs. After each epoch, the model is evaluated with three datasets: training, validation and testing datasets.

5.7 Results and Discussion

To analyze the performance of the proposed three models, CNN, LSTM and CNN-LSTM, the results from the ten folds are concatenated to form one data set of predicted scores. As a result, we have three datasets of predicted scores, one for every model. Then, five evaluation metrics are used to evaluate the predicted scores:

1. Recall or Sensitivity or TPR (True Positive Rate): Number of items correctly identified as positive out of total true positives.

$$Recall = \frac{TP}{TP + FN} \tag{5.1}$$

2. Precision: Number of items correctly identified as positive out of total items identified as positive.

$$Precision = \frac{TP}{TP + FP} \tag{5.2}$$

   To use these matrices, first we round the predicted probabilities to integer numbers. Then, the formulas of Precision and Recall are applied, Equations 5.1 and 5.2. These metric functions compare between the true class values and the predicted class values.

3. F1 Score: To see the balance between Precision and Recall, we use the F1 score, as shown in Equation 5.3. An F1 of "1" means a perfect score, while a score of "0" means a total failure.

$$F1 = 2 * \frac{Precision * Recall}{Precision + Recall} \tag{5.3}$$

4. ROC Curve: The predicted probabilities are used directly with the ROC curve to evaluate the model on different threshold values. The essential score with the ROC is the Area Under the Curve (AUC). The closer the AUC to 1, the better the performance of a model.

5. Confusion matrix or an error matrix [82] is a table that describes the performance of an algorithm or a classification model on a set of test data, typically supervised learning. All correct predictions are located in the diagonal of the table that includes information about true positive and true negative samples, Table 5.1. In our experiments, we use 0.5 as threshold value to evaluate the predicted scores and draw the confusion matrices. Also, the above metrics: Precision, Recall, F1 score, and Accuracy are calculated based on 0.5 threshold value.

Table 5.1: Confusion Matrix

	Predicted: Insider	**Predicted: Normal**
Actual: Insider	True Positive	False Negative
Actual: Normal	False Positive	True Negative

The next sections present our findings of using the aforementioned evaluation metrics.

5.7.1 Confusion Matrix of CNN Model

In this section, we present the using of confusion matrix to evaluate the results of CNN model. The predicted scores of testing datasets are evaluated using confusion matrix, as shown in Table 5.2. The CNN model shows a good performance with the testing dataset by predicting 49 of the malicious samples correctly and mispredicting 20 samples. On the other side, the model predicts 5553 of the negative samples correctly and mispredicts 1965 samples. Even though the number of positive examples are too small compared to the number of negative samples, the CNN model is able to distinguish between them in most cases.

With the testing dataset, described in Table 5.2, we can see that the model can detect 71% of the insider threat samples correctly, while falsely classify 26% of normal samples.

Table 5.2: CNN Confusion Matrix of all test cases. The true positive is 49, and the false positive is 1965. On the other side, the true negative is 5553 and false negative is 20.

N = 7587	**Predicted: Insider**	**Predicted: Normal**
Actual: Insider	49	20
Actual: Normal	1965	5553

5.7.2 CONFUSION MATRIX OF LSTM MODEL

In this section, we present the using of confusion matrix to evaluate the results of LSTM model. The confusion matrix of testing datasets are shown in Table 5.3. The LSTM model, with the testing datasets, predicts all malicious samples correctly; that means the true positive is 1 . On the other side, the model predicts 4712 of the negative samples correctly and mispredicts 2806 samples. The LSTM model performs well with the malicious examples. However, the false positive rate is 37%. The false positive rate is high compared to CNN and CNN-LSTM models.

Table 5.3: LSTM Confusion Matrix of all test cases. The true positive is 69, and the false positive is 2806. On the other side, the true negative is 4712 and false negative is 0.

N = 7587	**Predicted: Insider**	**Predicted: Normal**
Actual: Insider	69	0
Actual: Normal	2806	4712

5.7.3 CONFUSION MATRIX OF CNN-LSTM MODEL

In this section, a confusion matrix is used to evaluate the results of CNN-LSTM model. The predicted results from ten folds are concatenated to form one dataset. The formed dataset is evaluated using confusion matrix, as shown in Table 5.4. With the testing dataset, the CNN-LSTM predicts 58 of the malicious samples correctly and mispredicts 11 samples. On the other side, the model predicts 6189 of the negative samples correctly and mispredicts 1329 samples. The behavior of CNN-LSTM model is balance compared to CNN and LSTM models. The true positive rate of CNN-LSTM model is high compared to CNN model, while the true negative is 82% , which is a high rate compared to both CNN and LSTM models. Moreover, the false positive rate is 18% which is low compared to both CNN and LSTM models

The elapsed time to train CNN-LSTM model is also balanced compared to the other

models. The training time of CNN-LSTM is less than LSTM model, and greater than CNN model.

Table 5.4: CNN-LSTM Confusion Matrix of all test cases. The true positive is 58, and the false positive is 1329. On the other side, the true negative is 6189 and false negative is 11.

N = 7587	**Predicted: Insider**	**Predicted: Normal**
Actual: Insider	58	11
Actual: Normal	1329	6189

5.7.4 Performance Comparison in term of Precision, Recall, F1 Score, and AUC

In this section, we present extra metrics to evaluate the proposed models. We use precision, recall, and F1 score metrics to analyze the true and fasle rates of a model. The traditional way of using the accuracy metric alone is not sufficient because it evaluates the models on all data classes regardless of the distribution of these classes. In almost all insider misuse situations, the data is extremely imbalanced [6]. As illustrated in Table 5.3, the number of test cases is 7587. The positive samples are just 69, which means just 0.91% of test cases are insiders. So, we use precision and recall metrics to deeply investigate the performance of the models on the minority group.

The precision and recall analyze the results by considering just true positive, false positive, and false negative values. That is, we consider the predicted true positive samples and how many false positive and negative samples have also been predicted.

It is important to mention that the Precision, Recall, F1 and Accuracy metrics are calculated based on choosing 0.5 threshold value. Changing threshold value leads to different results; that is why we use ROC metric with AUC score to study broadly the behavior of the proposed models. ROC curve uses a list of threshold values ranging between 0 and 1.

The CNN model has a better performance in terms of accuracy compared to the LSTM

model, see Table 5.5. The accuracy metric is 0.74. The CNN model predicts 74% of total samples correctly. However, the distributions of the predicted scores are different between the normal and malicious classes. Thus, we need to investigate the results in more depth.

The recall of the CNN model is 0.68, which means 68% of the minority group are predicted correctly. However, the precision metric seems not good. The CNN model has 2.4% precision, and the LSTM and CNN-LSTM models have 2.4%, and 4.2% respectively. The reason behind the small precision values is the fact that the insider's data sets are imbalanced. By considering the CNN model, see Table 5.2, we can see that the model mispredicts 26% of the majority class, or the false positive is 1965. However, the precision is 2.4% because the minority class has just 69 samples. Even if the model predicted all the samples of the minority class correctly and mispredicted a small portion of the majority class, the precision values would be small.

There are other reasons that may lead to such behavior:

1. The nature of synthetic data samples.
2. The noise from augmentation operation during training phase.
3. The number of training samples.
4. The nature of imbalanced data.

The LSTM model presents the best performance in terms of detecting malicious insiders. We can see that clearly by just monitoring the Recall value, see Table 5.5, which is 1. This optimum Recall value, where the false negative rate equals zero, means all malicious samples are predicted correctly. However, the Precision value is small (0.024) which means we have a high false positive rate. 37% of normal samples are predicted as abnormal. This issue also leads to low accuracy value, 0.63.

The CNN-LSTM model shows a balanced behavior compared to the other two models. The CNN-LSTM model evaluates 84% of malicious samples correctly, while 17.6% of normal samples are falsely classified.

Table 5.5: Performance comparison of deep-learning based models. The results are presented in term of Precision, Recall, F1, Accuracy and AUC for all models. The CNN model shows better Accuracy score compared to the LSTM model and shows the lowest scores of Precision, Recall, and AUC. The LSTM model shows the best Recall and low Accuracy values, while CNN-LSTM model obtains the best F1 score.

Model	Precision	Recall	F1 Score	Accuracy	AUC	Epochs
CNN	0.024	0.71	0.047	0.74	0.85	10
LSTM	0.024	1	0.047	0.63	0.873	10
CNN-LSTM	0.041	0.84	0.08	0.82	0.862	10

To see the relation of both false positive and false negative samples, F1 score , Table 5.5, presents the weighted average of Precision and Recall of each models. The CNN-LSTM model achieves the best performance in term of F1 score.

5.7.5 VISUAL-BASED EVALUATION (ROC CURVE)

Receiver Operating Characteristic (ROC) curve is an essential tool to evaluate the predicted scores. In a ROC curve, the true positive rate (Sensitivity (Recall)) is plotted against the false positive rate (1-Specificity) for different cut-off points. The area under the ROC curve (AUC) is a measure of how well a model can distinguish between two distribution (normal/malicious) behaviors.

The AUC results of the deep-based models are presented in Table 5.5. For every model, the predicted probability scores from ten folds are concatenated and evaluated using the area under the curve (AUC). The CNN model shows AUC score of 0.85, while LSTM and CNN-LSTM show 0.873, and 0.862 respectively. The ROC curves of testing datsets with AUC metrics are shown in Figure 5.9.

We develop our training procedure to use AUC metric. The training, validation, and testing datasets are evaluated after each epoch using AUC metric. The validation datasets are used to choose the best model.

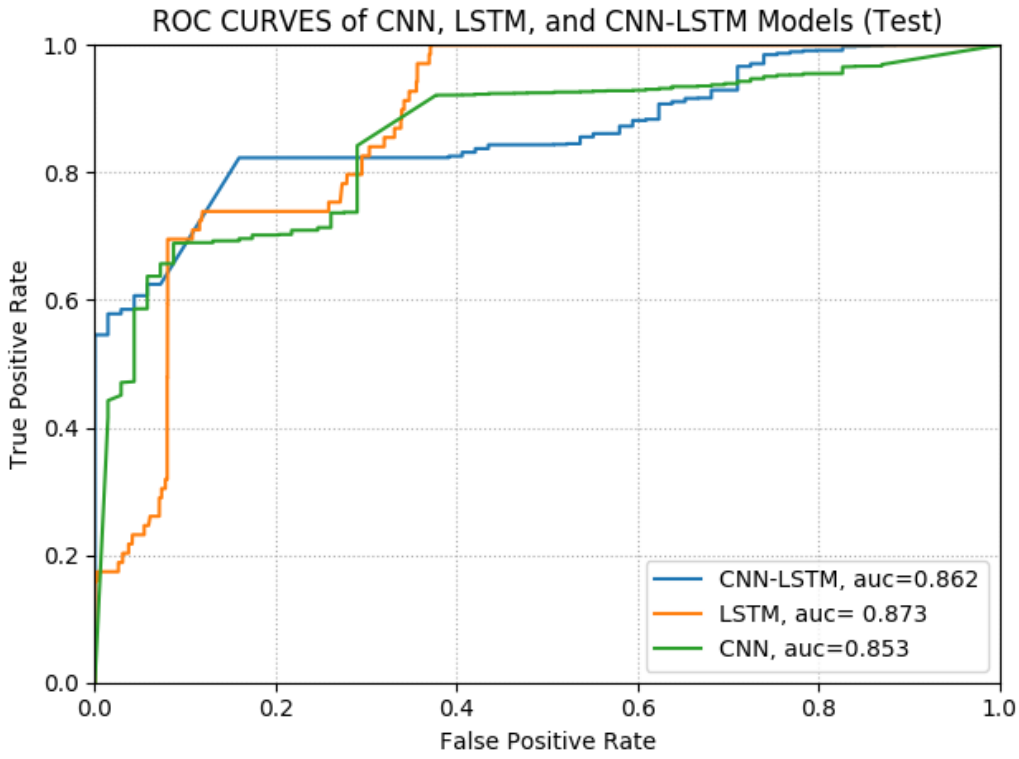


Figure 5.9: ROC curves of CNN, LSTM, and CNN-LSTM models with Area Under the Curve (AUCs). The models are evaluated using testing datasets.

5.8 Results Visualization

In this section, we visualize the results of the proposed models, CNN, LSTM , and CNN-LSTM. The histogram plots are used to draw the distribution of the predicted scores of testing datasets. We use 100 bar intervals in our histograms to clarify the distribution of the predicted scores.

A class coloring is used to visualize the results. This helps to see the separation between the two classes, normal and malicious users. However, the imbalanced distribution of insider and normal samples make the visualization hard, especially with testing datasets.

5.8.1 Visualization of CNN Results

The predicted probability scores of testing datasets with the CNN model are visualized using histogram plot as shown in Figure 5.10. The used testing or validation datasets are

unbalanced, where the number of positive samples are just 0.91% of total testing samples. The unbalanced distripution of the normal and malicious samples makes the visualization of the minority group very difficult.

We can see in Figure 5.10 that there is an overlap between the predicted scores. Whenever the model falsely predicts a portion of the majority class, that leads to overlap in the histogram. Even a small portion of majority class leads to a big overlap. Thus, using the tradition scales with histograms makes the visualization of positive samples very hard.

To optimize the presentation of the histograms, we use the log scale of base 10. The log scale histogram allows high-frequency ranges to be displayed without low-frequency ranges being squeezed down into the bottom of the graph.

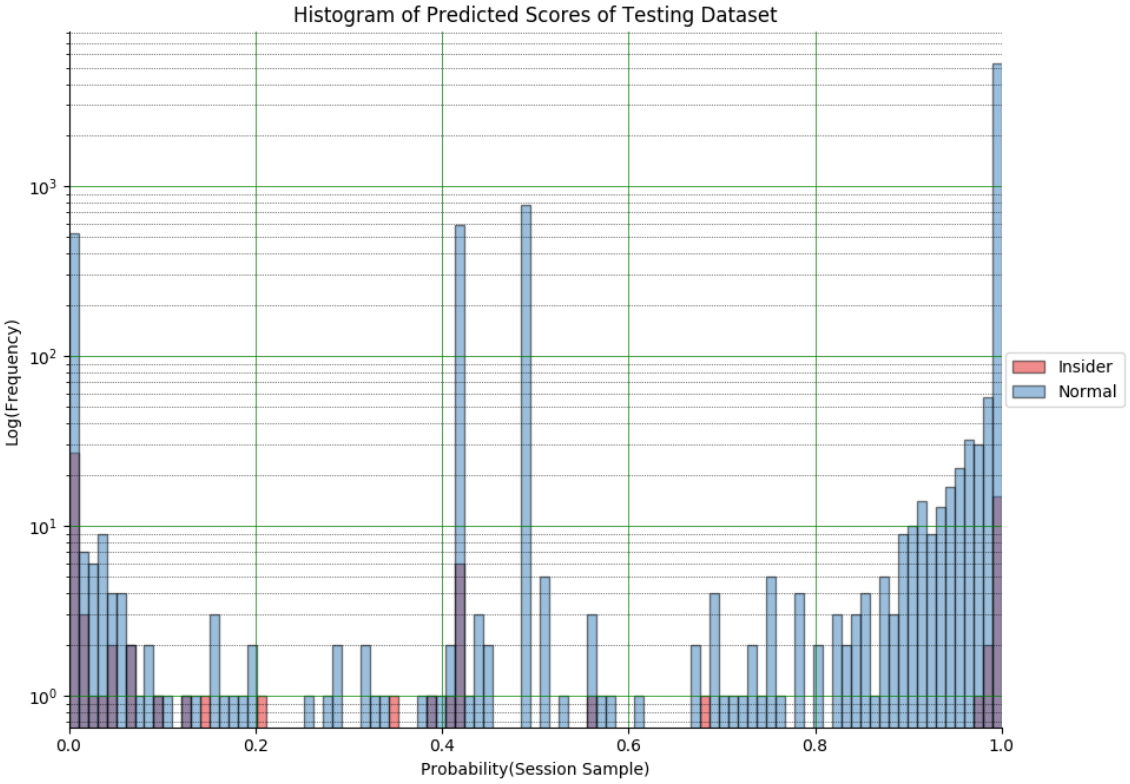


Figure 5.10: Histogram of CNN predicted scores of testing dataset.

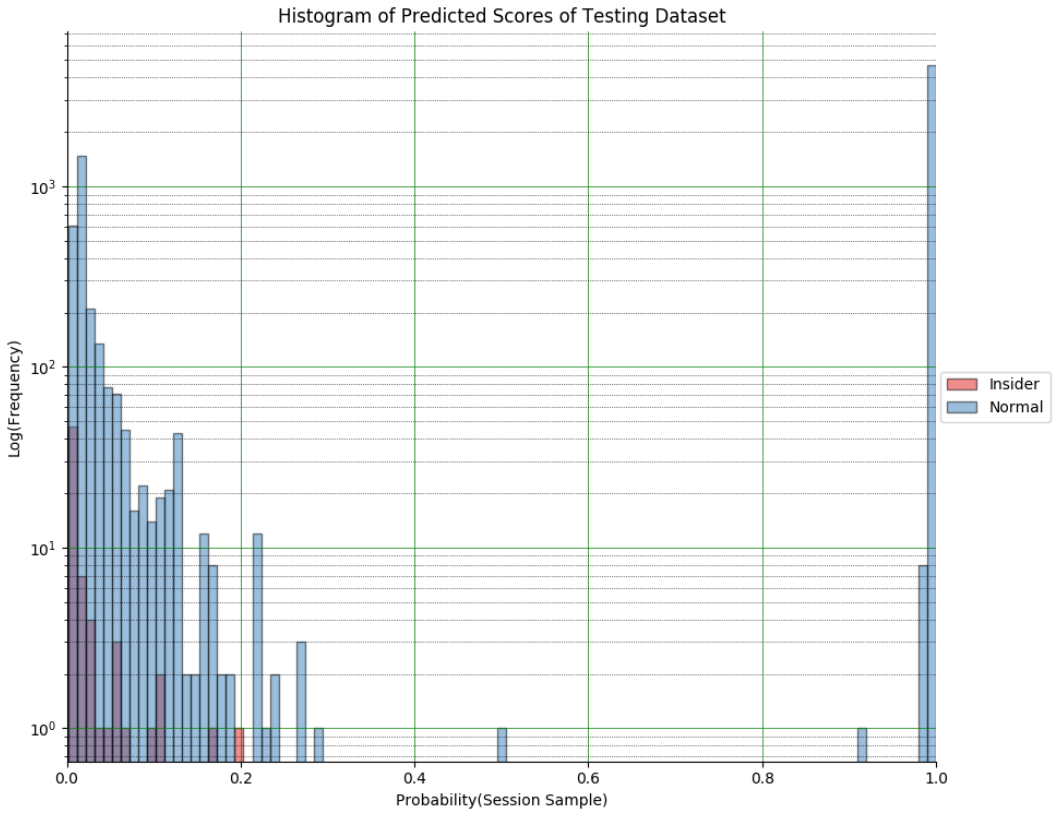


Figure 5.11: Histogram of LSTM predicted scores of testing dataset.

5.8.2 Visualization of LSTM results

The predicted probability scores of testing datasets with LSTM model are visualized using a histogram plot as shown in Figure 5.11. We can clearly see how all malicious samples (red color) are detected correctly by the LSTM mode. However, many of the normal samples are falsely classified as insiders.

5.8.3 Visualization of CNN-LSTM results

The predicted probability scores of testing datasets with CNN-LSTM model are visualized using a histogram plot as shown in Figure 5.12. The CNN-LSTM model shows a good separation between the normal and malicious samples. However, there is overlap on the two edges of the histograms and on one bar in the middle that represent the misclassification of data samples.

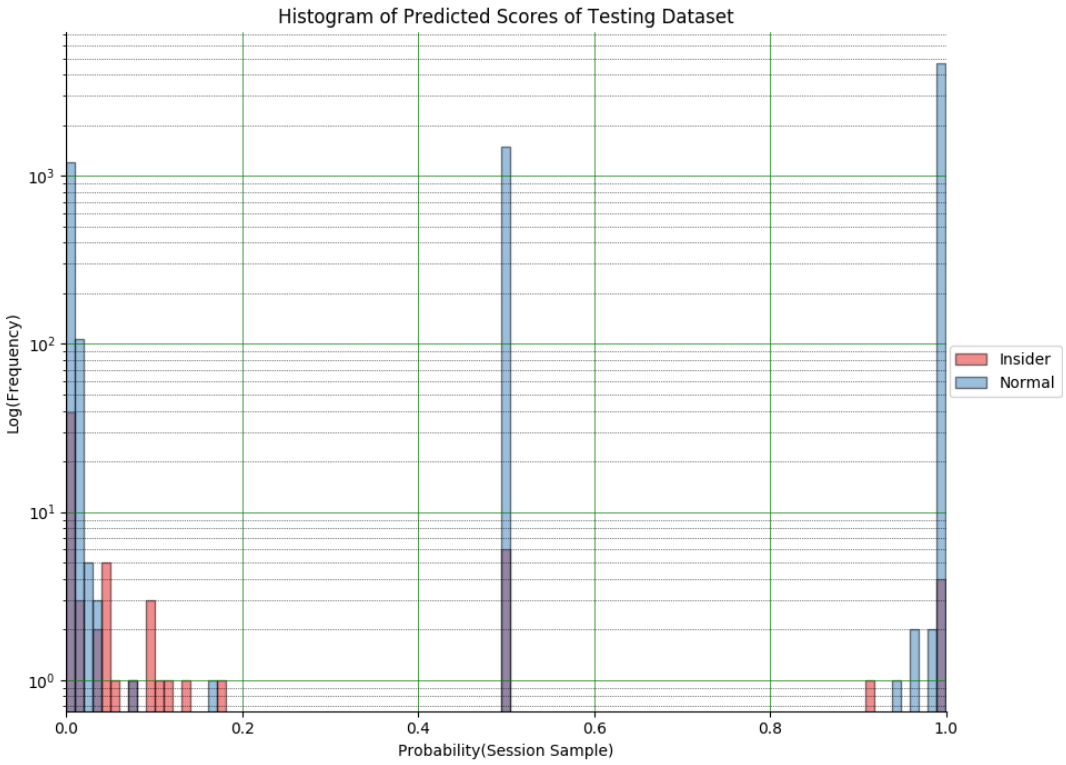


Figure 5.12: Histogram of CNN-LSTM predicted scores of testing dataset.

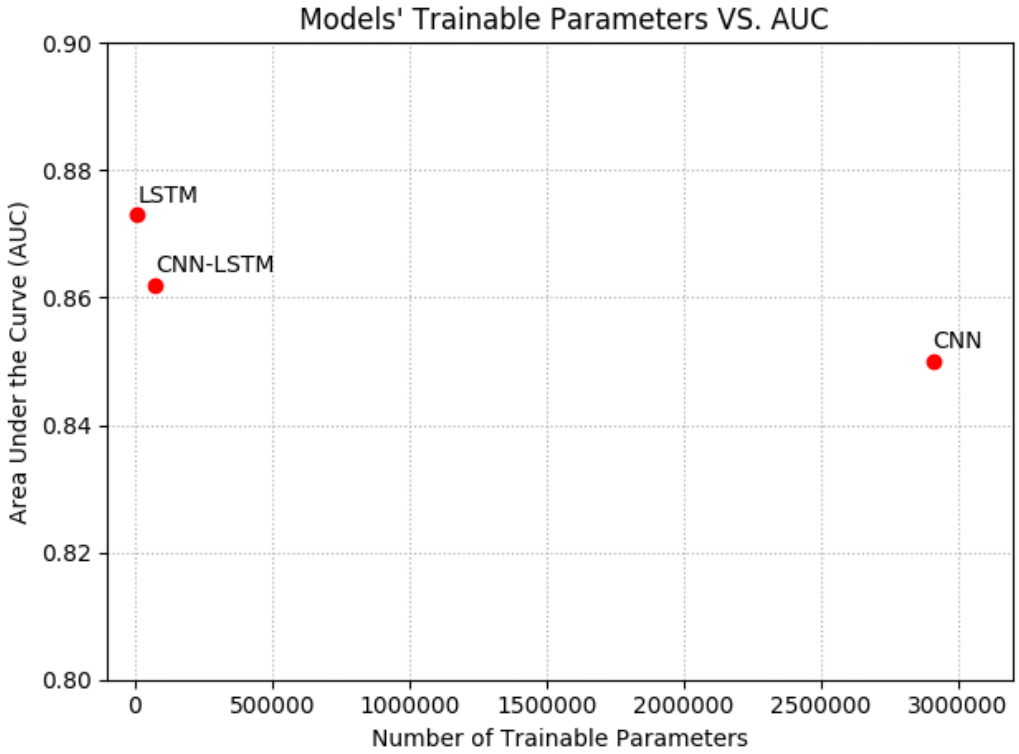


Figure 5.13: Deep-based models' trainable parameters Vs. Area Under the Curve (AUC). The trainable parameters of the fully connected layer are included.

5.9 Model Complexity

In the evaluation approach, we also consider the complexity of models' architectures by examining the number of trainable parameters of each model, see Figure 5.13. The trainable parameters of CNN model are doubled 10 times after DNN layer. This increase in parameter number is due to the flatten layer. We flatten 32 feature maps of size 9032 and connect them to DNN layer. There is no flatten operation with the other models. Thus, we see a huge difference of trainable parameters.

5.10 Learning Curves

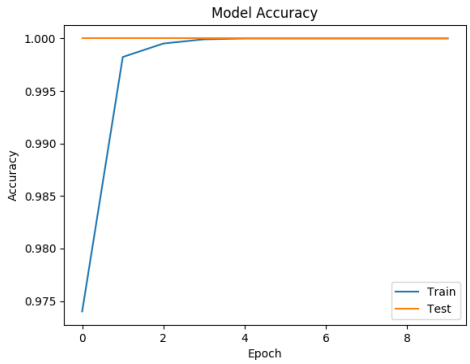
In this section, we present accuracy and loss curves during training. As a case study, the work is focused on fold 1. For this study, we use number of epochs equal 10, and the splitting ratio between the training and validation sets is 0.1. Figure 5.14 (a, b, c, d) shows the loss and accuracy curves, where the orange curve represents the loss, and the blue curve represents the accuracy.

The loss indicates how well the model is doing for these two sets. The loss curve is an average of the errors made for all samples in every epoch. The optimization operation occurs every mini-batch of size 32, in both training and validation sets.

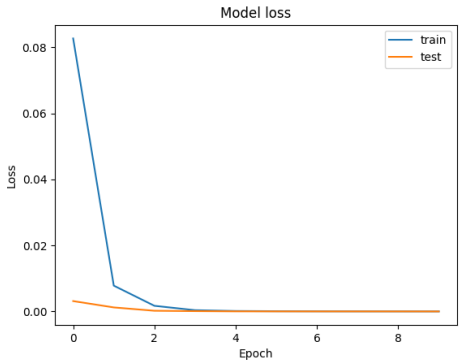
On the other side, the accuracy curve is the percentage of the correctly predicted samples to the total number of validation samples.

The main objective in a learning model is to minimize the loss function's value with respect to the model's parameters by changing the weight values.

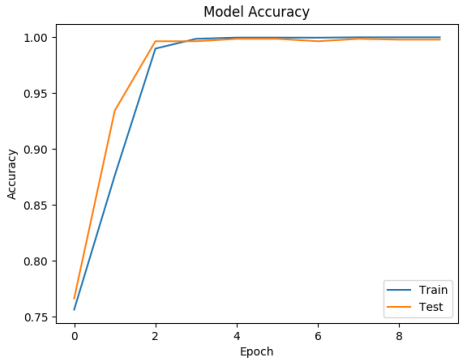
Figure 5.14 presents four trends instead of two because we do two experiments. In the first experiment, we get the accuracy and loss trends, as shown in Figure 5.14a And Figure 5.14b. The noticeable thing about these figures is that the validation accuracy is larger than the training accuracy.



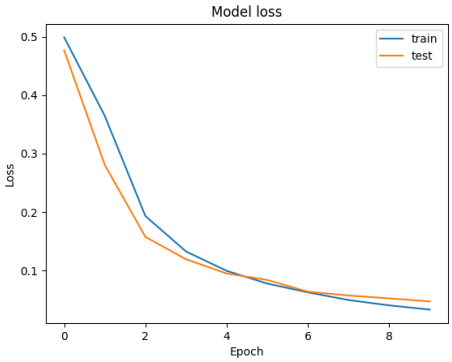
(a) Fold 1 Model Accuracy on Train and Validation Datasets before Shuffling



(b) Fold 1 Model Loss on Train and Validation Datasets before Shuffling



(c) Fold 1 Model Accuracy on Train and Validation Datasets after Shuffling



(d) Fold 1 Model Loss on Train and Validation Datasets after Shuffling

Figure 5.14: Accuracy and Loss VS. Epochs of CNN model.

Also, the validation loss is less than the training loss. Upon more investigation of the data, we find that the distribution of the validation dataset has one class. This issue appears because of two reasons:

1. Using of validation_split attribute: This attribute is used with the fit function in Keras. It determines the fraction of the training data to be used as validation data. This attribute helps to split the data by just giving the splitting ratio. However, the validation data is selected from the last samples of the provided data before shuffling. Thus, the data is split, shuffled, and then the model training starts.

2. SMOTE: we use SMOTE augmentation approach to keep the distributions of classes

balanced. The important observation here is that the SMOTE returns data without shuffling.

By combining the above two points, we conclude that the last part of the dataset has a one class distribution. This is exactly what we see in the validation dataset. Thus, the datasets should be shuffled before using validation_split attribute. By doing this shuffling, we make the distribution of the validation dataset balanced. Figure 5.14(c, d) shows the accuracy and loss trends of the model after shuffling the dataset. We can see clearly the effect of the shuffling operation on the accuracy and loss of validation data sets.

CHAPTER 6

COMPARING THE PERFORMANCE OF DIFFERENT DEEP NETWORK STRUCTURES ON DETECTING INSIDERS

Detecting of insider misuse is one of the most challenging tasks in cybersecurity research. Researchers study the insiders' problem from different perspectives. Some studies focus on technical solutions such as big data, statistical, or graph analytic methods. Others try to determine the user's intention via semantic or sentiment analysis of log data, such as email or instant messaging, or surfing web logs [41].

In this chapter, we develop our approaches, proposed in Chapter 5, by exploring different deep learning architectures for sentiment analysis on insider misuse detection.

6.1 LETTER-BASED DETECTION MODELS

We explore different structures of letter-based sentiment models for insiders' misuse detection. The aim of this work is to analyze the effects of building deeper or wider neural networks on the detection performance. The final results are evaluated using Precision, Recall, F1 score, and AUC metrics. In general, this work can be summarized as follows:

1. Studying the impact of increasing the number of CNN or LSTM layers on the model performance.

2. Analyzing the effects of increasing the number of nodes (hidden units) per layer on the model performance.

3. Studying the effect of using regularization techniques such as dropout, batch normalization, and L2 norm.

We use a 10-fold cross-validation procedure to train and evaluate the proposed models using the same data points in Chapter 5. We split the datasets into ten groups where each group includes independent subjects. That is, within every fold, one group is set for validation, one group is set for testing, and the rest of the groups are concatenated to form a training dataset. Thus, the models are trained and evaluated on different training, validation, and testing datasets.

6.2 Regularization for Deep Learning Networks

Regularization is a crucial term to describe the techniques that are used to deal with overfitting problems. Regularization strategies aim to generalize the performance of the machine learning models by helping them perform well on unseen data. In this section, we present two essential regularization strategies that are used to develop different-structure deep neural networks:

1. Dropout
2. Batch normalization
3. L2 norm

6.2.1 Dropout

The dropout technique is used with deep-learning models to reduce the overfitting and improve their generalization. The dropout technique is applied during the training phase by randomly disabling neuron connections. The disabling of neurons is done by setting the neuron connections' weights to zero. Intuitively, setting the connection weights to zero prevents layers from learning from the input behind these connections. Thus, these

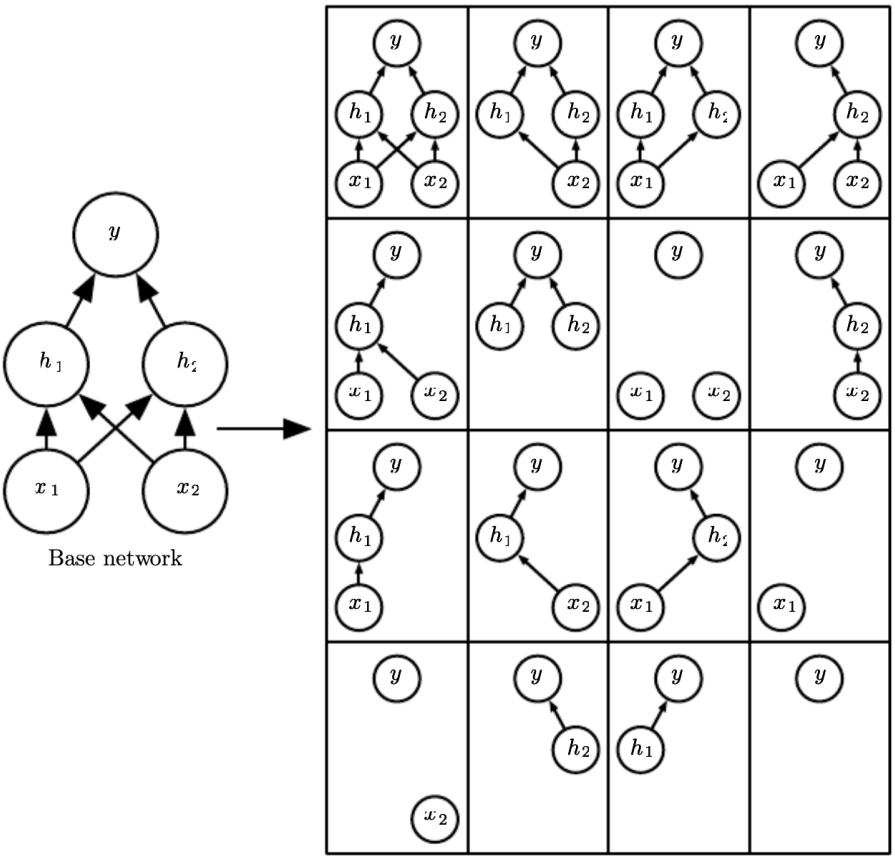


Figure 6.1: Dropout strategy. On the left is the base network. There are two visible units (input and output) and two hidden units. Training the base network using dropout would result in sixteen possible cases. This strategy is applied by excluding nonoutput units from an underlying base network, as shown in the right figure [32].

layers will rely more on other inputs, and that helps them to be more generalized [32]. The regularization process of using dropout is shown in Figure 6.1 .

6.2.2 BATCH NORM

Batch normalization regularizers ensure that the deep learning models produce an output with a specific distribution. Thus, the normalization layers are set after CNN, LSTM, or even DNN layers but before the activation functions [42].

Batch normalization is used to reparametrize deep learning models. The essential assumption of training deep models is that the gradient algorithm changes the parameters

within each layer without affecting other layers. However, the update occurs in all layers simultaneously, which may lead to undesired results [32].

The Batch normalization reduces the problem of parameters updating ("covariate shift") across different layers. Covariate shift points to the variation in the distribution of the input values to a learning algorithm [42]. The reparametrization process can be applied to any input or hidden layer in a network. For a minibatch H of size m samples, H can be normalized as follows [32]:

$$\mu = \frac{1}{m}\sum_{i}^{m} H_i \tag{6.1}$$

$$\sigma = \sqrt{\delta + \frac{1}{m}\sum_{i}^{m} (H-\mu)_i^2} \tag{6.2}$$

$$H' = \frac{H-\mu}{\sigma} \tag{6.3}$$

where μ, σ are vectors containing the mean and standard deviation of each unit. $\delta > 0$ is a small scalar to avoid the undefined number $\sqrt{0}$.

6.2.3 L2 NORM

The L2 norm penalty is commonly known as weight decay or Ridge regression. It sums the "squared magnitude" of coefficient as penalty term to the loss function as shown in Equation 6.4, and Equation 6.5. Adding the penalty term to the objective function makes the weights closer to the origin point [32].

$$||w||_2 = (|w|_1^2 + |w|_2^2 + ... + |w|_n^2)^{1/2} \tag{6.4}$$

$$Loss = Error(y, \hat{y}) + \lambda \sum_{i=1}^{n} w_i^2 \tag{6.5}$$

6.3 Exploring of Deep Learning Architectures

In this chapter, we develop four different deep learning architectures compared to our base model proposed in Chapter 5: CNN, LSTM, and CNN-LSTM. Regularization strategies are applied to generalize models' performance. The rationale of each architecture can be summarized as follows:

1. Architecture one: we increase the number of hidden units per layer. For example, using 64 filters with CNN layer instead of 32.

2. Architecture two: For each base layer, we add one subsequent layer. For example, using two CNN layers.

3. Architecture three: For each base layer, we add two subsequent layers. For example, using three CNN layers.

4. Architecture four: For each base layer, we add one subsequent layer and increase the number of hidden units. For example, with CNN model, we use two subsequent CNN layers with 64 filters.

The first layer (the input or representation layer) in all models is the same, where the input textual session-based sequence is converted into a matrix of numbers.

The input text is converted into a list of tokens, or a list of characters. Then, the list of tokens is converted into a list of the character index. Then, a list of characters-index is used to build a dictionary. The character index is just the location number of that character in the dictionary of unique characters that occur in the entire sessions.

To solve the problem of invariant session lengths, each input session is padded with zeros to make all session lengths equal to the longest session in the dataset. The output of this layer would be a vector with the length equal to the longest session. The keras library [19] is used to obtain the vocabulary of character indexes and to pad the input

session to get the same fixed length. The output of this layer is a 2d matrix. Each row is a 100-D character vector representing each character in the input session.

The next sections present the use of the above architectures with the CNN, LSTM, and CNN-LSTM models.

6.4 CNN-BASED MODELS

In this section, we present the use of CNN layer with four different architectures. We aim to study the models' performances on insider's misuse detection under these different architectures.

6.4.1 CNN ARCHITECTURE ONE

The first model is a one-layer CNN with 64 filters followed by two densely connected (DNN) layers with ten, and one hidden units respectively. "relu" and "sigmoid" activation functions are used with CNN and DNN respectively, see Figure 6.2. In this model, we utilized character embeddings to represent the input session sequence. We tried to let the model learn these embeddings by considering them as another hyper-parameter that the model needs to learn. The hyper-parameters of this model are: embedding size: 100, CNN filters: 64, filter size: 3, pool size: 2, batch size: 32, epochs: 20, dropout: 0.1, and L2: 0.01.

A batch normalization layer is used with CNN layer to contain a covariate shift by normalizing the activations of each layer. This, probably, supports each layer to learn on a more stable distribution of inputs. Thus, batch normalization could accelerate the training of the networks.

The Adam optimization method is used to update the model's weights. The optimization process is applied after every 32 samples. The average error over these 32 samples is propagated back using the Adam algorithm. The learning rate is reduced over time using weight decay or L2: 0.1.

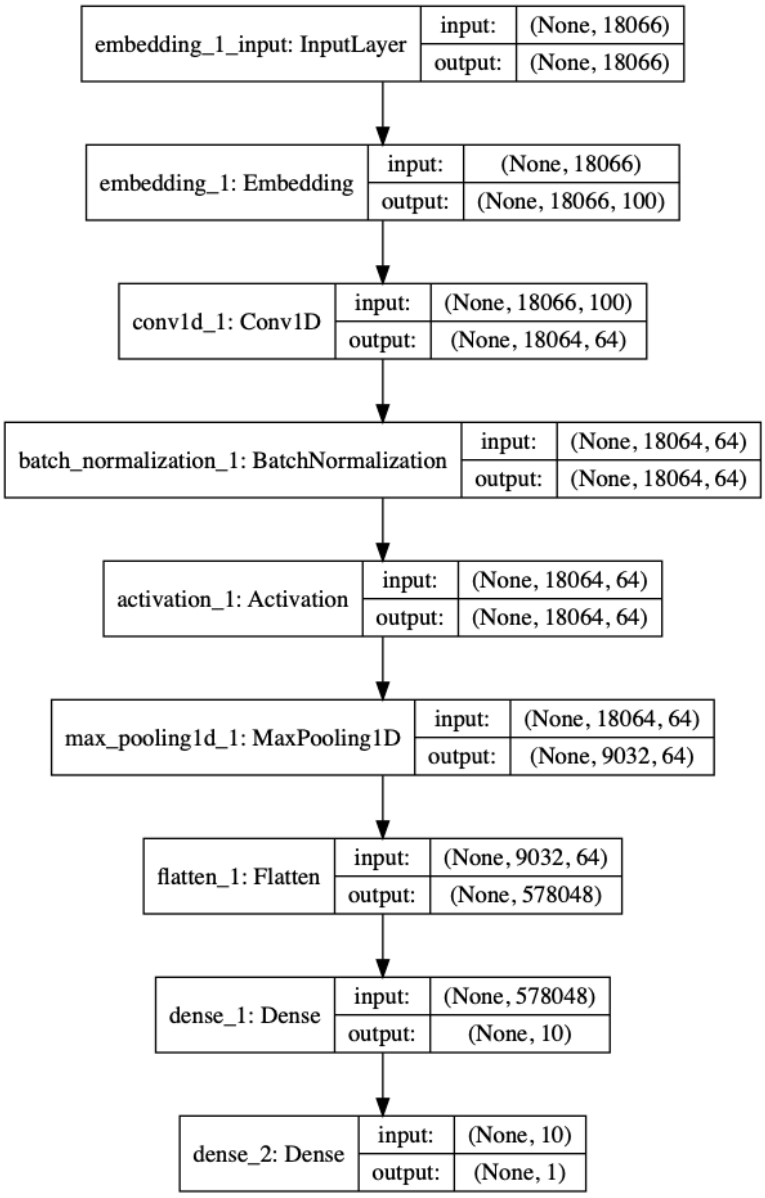


Figure 6.2: CNN model (Architecture 1).

6.4.2 CNN Architecture two

The second model includes two consecutive CNN layers with 64 filters, followed by two DNN layers with ten, and one hidden units. “relu” and “sigmoid” activation functions are used with CNN and DNN respectively, see Figure 6.3. The hyper-parameters of this model are: embedding size: 100, CNN filters: 64, filter size: 3, pool size: 2, batch size: 32, epochs: 20, dropout: 0.1, and L2: 0.01.

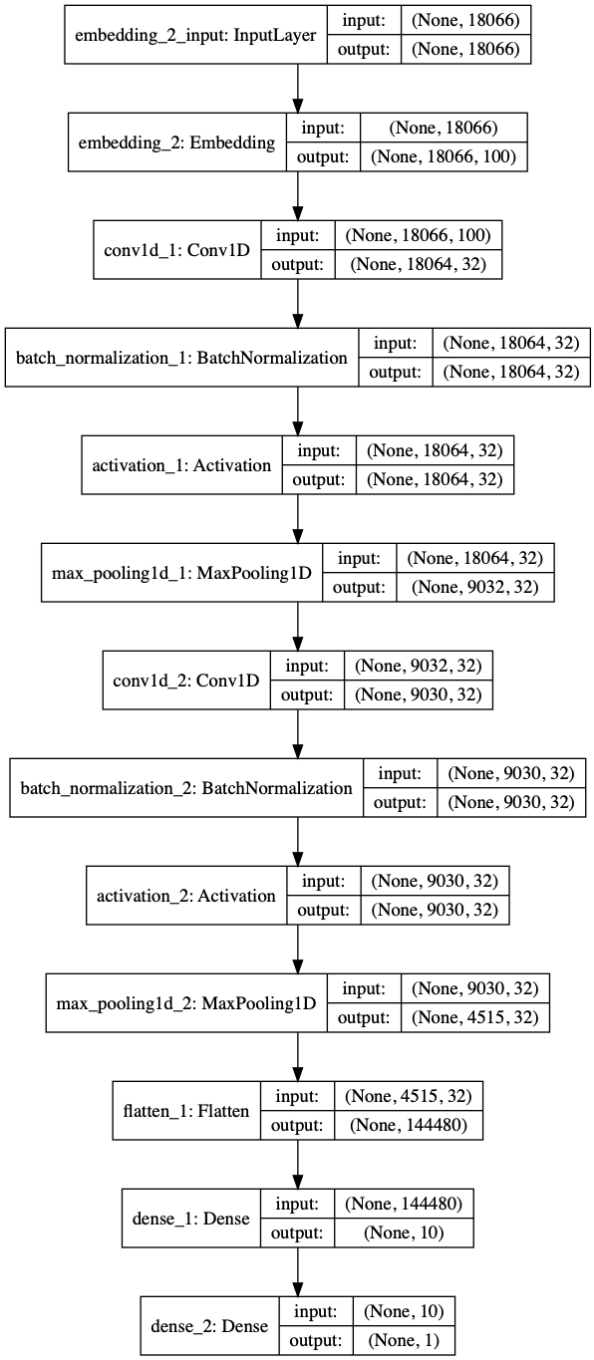


Figure 6.3: CNN model (Architecture 2).

The main difference of this model compared to the first one is the number of feature maps. In architecture two, we consider more textual features by using 64 filters.

6.4.3 CNN ARCHITECTURE 3

The third model includes three consecutive CNN layers with 32 filters, followed by two DNN layers with ten, and one hidden units. “relu” and “sigmoid” activation functions are used with CNN and DNN respectively, see Figure 6.4.

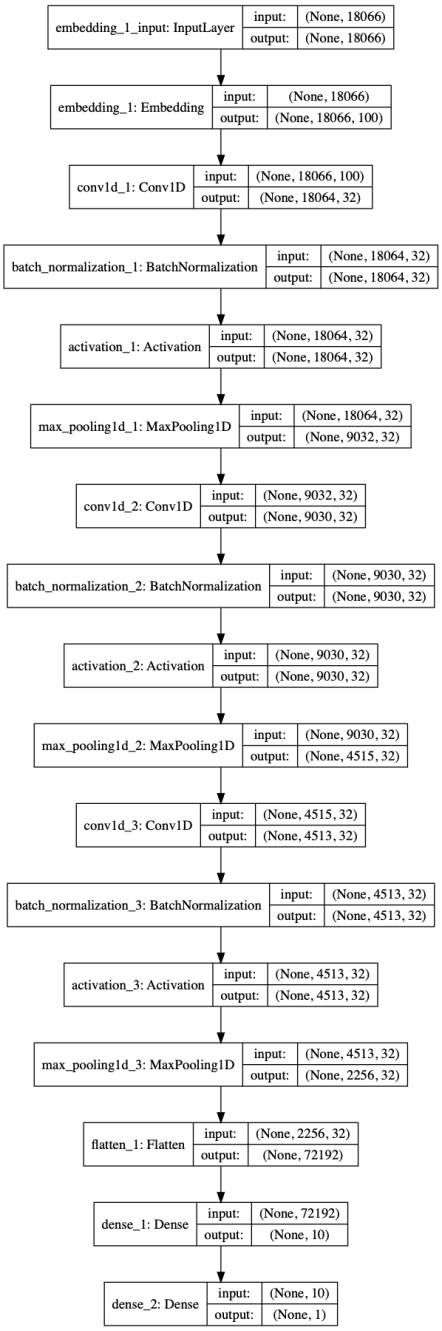


Figure 6.4: CNN model (Architecture 3).

The hyper-parameters of this model are: embedding size: 100, CNN filters: 32, filter size: 3, pool size: 2, batch size: 32, epochs: 20, dropout: 0.1, and L2: 0.01. This architecture explores adding extra CNN layers compared to architecture two. One noticeable thing about going deep is the cut of the trainable parameters after each CNN layer.

6.4.4 CNN ARCHITECTURE 4

The fourth architecture includes adding two consecutive CNN layers with 64 filters, followed by two DNN layers with ten, and one hidden units. “relu” and “sigmoid” activation functions are used with CNN and DNN respectively, see Figure 6.5.

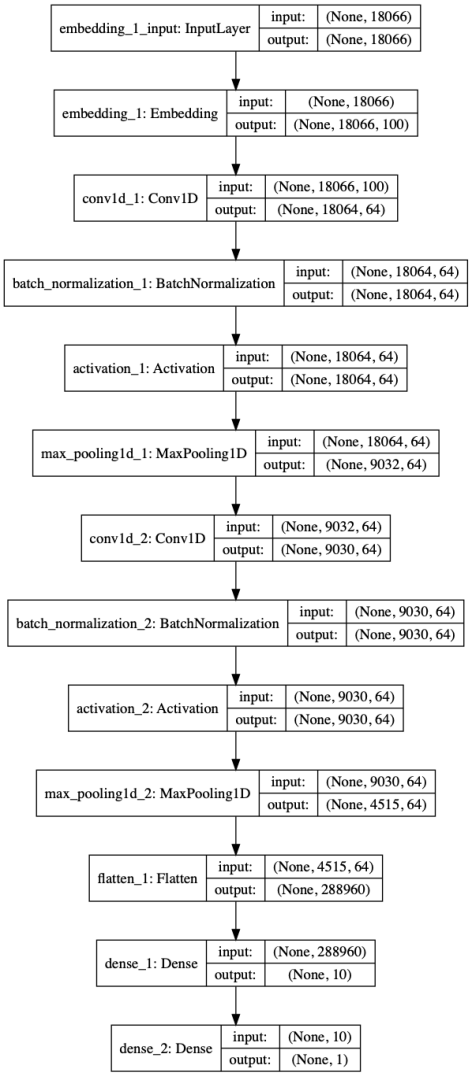


Figure 6.5: CNN model (Architecture 4).

6.5 LSTM-BASED MODELS

In this section, we present the use of LSTM layer with four different architectures. We develop these architectures using LSTM layer with various regularization techniques.

6.5.1 LSTM ARCHITECTURE ONE

The first model is a one-layer LSTM with 200 hidden units followed by two densely connected (DNN) layers with ten, and one hidden units respectively, see Figure 6.6. "relu" and "sigmoid" activation functions are used with CNN and DNN respectively. In this model, we utilize character embeddings to represent the input session sequence. We try to let the model learn these embeddings by considering them as another hyper-parameter that the model needs to learn. The hyper-parameters of this model are: embedding size: 100, batch size: 32, epochs: 20, dropout: 0.1, recurrent dropout: 0.1, and L2: 0.01.

A batch normalization is used with LSTM layer to contain a covariate shift by normalizing the activations of the layer. Moreover, L2 norm is used to reduce the learning rate over time.

The Adam optimization method is used to update the model's weights. The optimization process is applied after every 32 samples. The average error over these 32 samples is propagated back using the Adam algorithm.

6.5.2 LSTM ARCHITECTURE TWO

The second architecture is developed using two-layer LSTM with 100 hidden units followed by two densely connected (DNN) layers with ten, and one hidden units respectively, see Figure 6.7. "relu" and "sigmoid" activation functions are used with CNN and DNN respectively. In this model, we utilize character embeddings to represent the input session sequence. We try to let the model learn these embeddings by considering them as another hyper-parameter that the model needs to learn. The hyper-parameters of this model are:

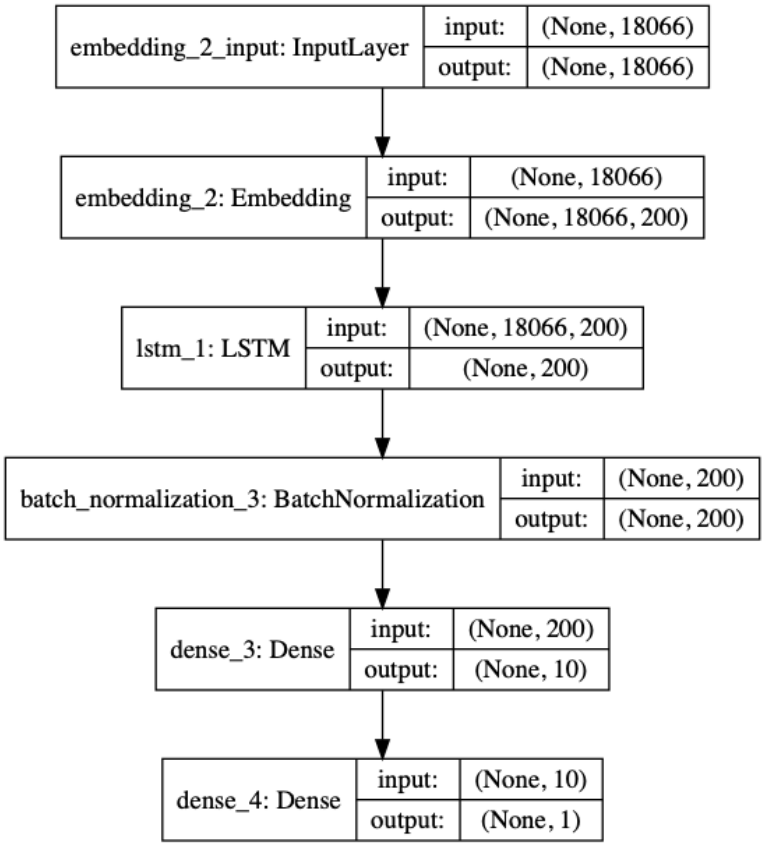


Figure 6.6: LSTM model (Architecture 1).

embedding size: 100, batch size: 32, epochs: 20, dropout: 0.1, recurrent dropout: 0.1, and L2: 0.01.

A batch normalization is used with LSTM layers to contain a covariate shift by normalizing the activations of LSTM layers. Moreover, L2 norm is used to reduce the learning rate over time.

The Adam optimization method is used to update the model's weights. The optimization process is applied after every 32 samples. The average error over these 32 samples is propagated back using the Adam algorithm.

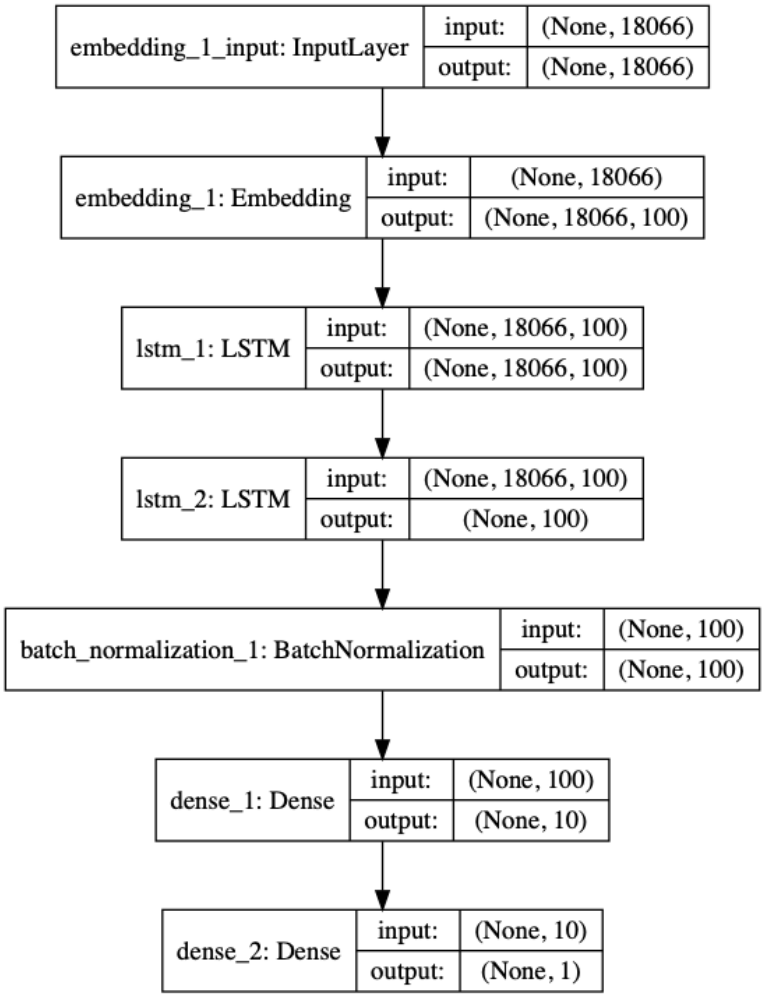


Figure 6.7: LSTM model (Architecture 2).

6.5.3 LSTM Architecture Three

The third architecture is developed using three LSTM layers with 100 hidden units followed by two densely connected (DNN) layers with ten, and one hidden units respectively, see Figure 6.8. "relu" and "sigmoid" activation functions are used with CNN and DNN respectively. In this model, we utilize character embeddings to represent the input session sequence. We try to let the model learn these embeddings by considering them as another hyper-parameter that the model needs to learn. The hyper-parameters of this model are: embedding size: 100, batch size: 32, epochs: 20, dropout: 0.1, recurrent dropout: 0.1, and L2: 0.01.

A batch normalization is used with LSTM layers to contain a covariate shift by normalizing the activations of LSTM layers. Moreover, L2 norm is used to reduce the learning rate over time.

The Adam optimization method is used to update the model's weights. The optimization process is applied after every 32 samples. The average error over these 32 samples is propagated back using the Adam algorithm.

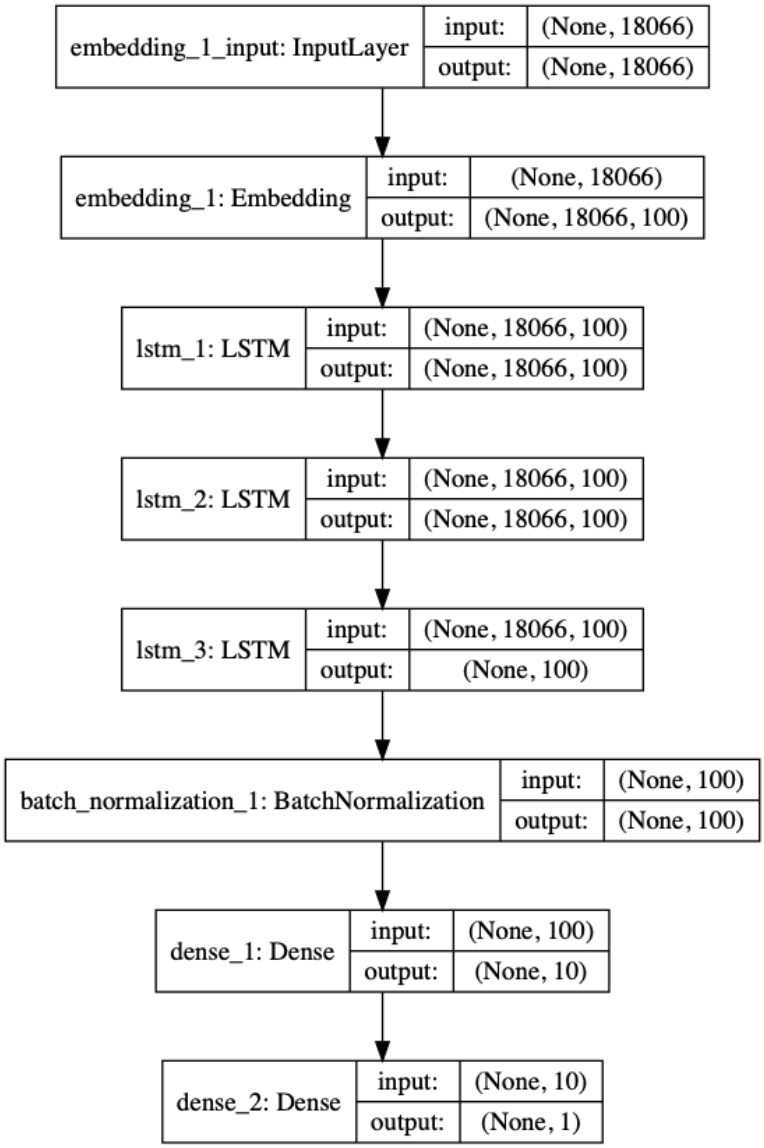


Figure 6.8: LSTM model (Architecture 3).

6.5.4 LSTM Architecture Four

The fourth architecture is developed using two LSTM layers with 200 hidden units followed by two densely connected (DNN) layers with ten, and one hidden units respectively, see Figure 6.9. "relu" and "sigmoid" activation functions are used with CNN and DNN respectively. In this model, we utilize character embeddings to represent the input session sequence. We try to let the model learn these embeddings by considering them as another hyper-parameter that the model needs to learn. The hyper-parameters of this model are: embedding size: 100, batch size: 32, epochs: 20, dropout: 0.1, recurrent dropout: 0.1, and L2: 0.01.

A batch normalization is used with LSTM layers to contain a covariate shift by normalizing the activations of LSTM layers. Moreover, L2 norm is used to reduce the learning rate over time.

The Adam optimization method is used to update the model's weights. The optimization process is applied after every 32 samples. The average error over these 32 samples is propagated back using the Adam algorithm.

6.6 CNN-LSTM-based Models

In this section, we present the use of both CNN and LSTM layers to develop four different architectures. We apply various regularization techniques that include drop out, batch normalization, and L2 norm.

6.6.1 CNN-LSTM Architecture one

We develop the first architecture using a CNN layer followed by an LSTM layer. We use 64 filters with the CNN layer, and 100 hidden units with the LSTM layer. The CNN-LSTM layer is followed by two densely connected (DNN) layers with ten, and one hidden units respectively, see Figure 6.10. "relu" activation functions are used with CNN and LSTM layers, while "sigmoid" functions are used with the DNN layers. In this model, we utilize

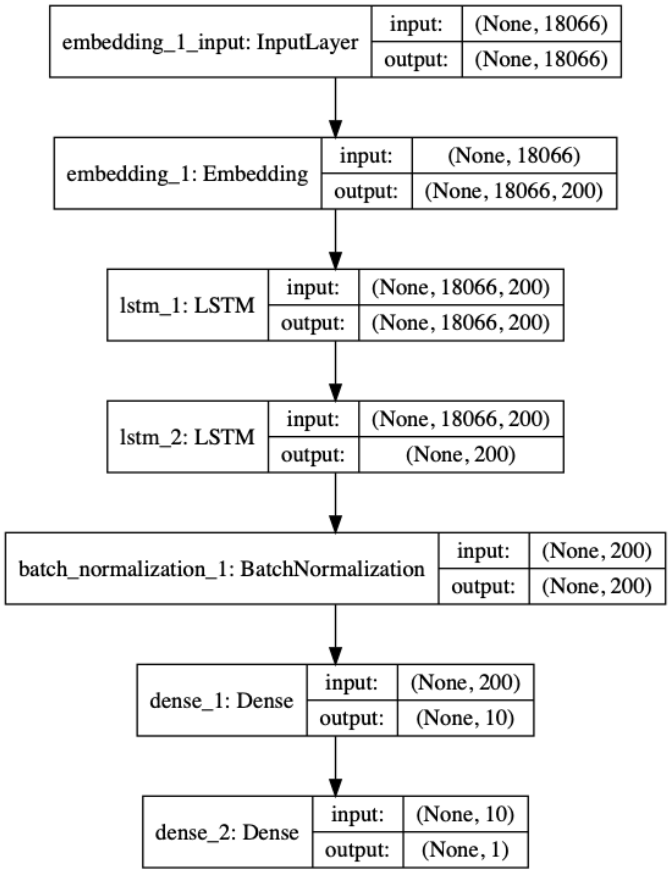


Figure 6.9: LSTM model (Architecture 4).

character embeddings to represent the input session sequence. We try to let the model learn these embeddings by considering them as another hyper-parameter that the model needs to learn. The hyper-parameters of this model are: embedding size: 100, CNN filters: 64, filter size: 3, pool size: 2, batch size: 32, epochs: 20, dropout: 0.1, recurrent dropout: 0.1, and L2: 0.01.

A batch normalization is used with LSTM layer to contain a covariate shift by normalizing the activations of the layer. Moreover, L2 norm is used to reduce the learning rate over time.

The Adam optimization method is used to update the model's weights. The optimization process is applied after every 32 samples. The average error over these 32 samples is propagated back using the Adam algorithm.

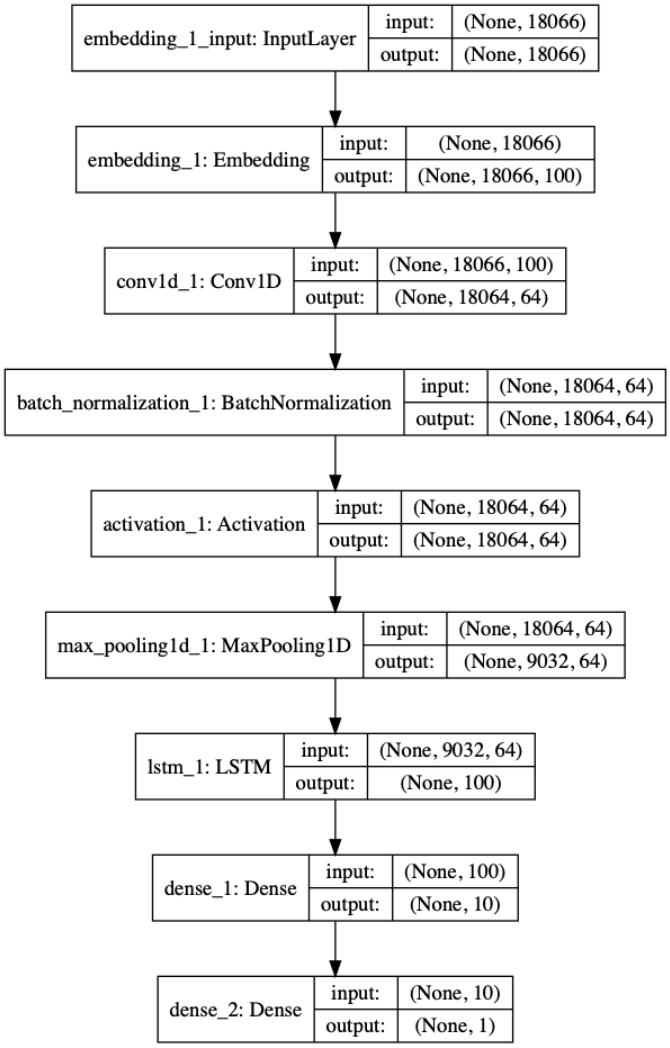


Figure 6.10: CNN-LSTM model (Architecture 1).

6.6.2 CNN-LSTM Architecture Two

We develop the second architecture using two CNN layers followed by an LSTM layer. We use 32 filters with the CNN layer, and 100 hidden units with the LSTM layer. The LSTM layer is followed by two densely connected (DNN) layers with ten, and one hidden units respectively, see Figure 6.11. "relu" activation functions are used with CNN and LSTM layers, while "sigmoid" functions are used with the DNN layers. In this model, we utilize character embeddings to represent the input session sequence. We try to let the model learn these embeddings by considering them as another hyper-parameter that the model needs to

learn. The hyper-parameters of this model are: embedding size: 100, CNN filters: 32, filter size: 3, pool size: 2, batch size: 32, epochs: 20, dropout: 0.1, recurrent dropout: 0.1, and L2: 0.01.

A batch normalization is used with CNN layers to contain a covariate shift by normalizing the activations of the layer. Moreover, L2 norm is used to reduce the learning rate over time.

The Adam optimization method is used to update the model's weights. The optimization process is applied after every 32 samples. The average error over these 32 samples is propagated back using the Adam algorithm.

6.6.3 CNN-LSTM Architecture Three

We develop the third architecture using two CNN layers followed by two LSTM layers. We use 32 filters with the CNN layer, and 100 hidden units with the LSTM layer. The LSTM layer is followed by two densely connected (DNN) layers with ten, and one hidden units respectively, see Figure 6.12. "relu" activation functions are used with CNN and LSTM layers, while "sigmoid" functions are used with the DNN layers. In this model, we utilized character embeddings to represent the input session sequence. We try to let the model learn these embeddings by considering them as another hyper-parameter that the model needs to learn. The hyper-parameters of this model are: embedding size: 100, CNN filters: 32, filter size: 3, pool size: 2, batch size: 32, epochs: 20, dropout: 0.1, recurrent dropout: 0.1, and L2: 0.01.

A batch normalization is used with CNN layers to contain a covariate shift by normalizing the activations of the layer. Moreover, L2 norm is used to reduce the learning rate over time.

The Adam optimization method is used to update the model's weights. The optimization process is applied after every 32 samples. The average error over these 32 samples is propagated back using the Adam algorithm.

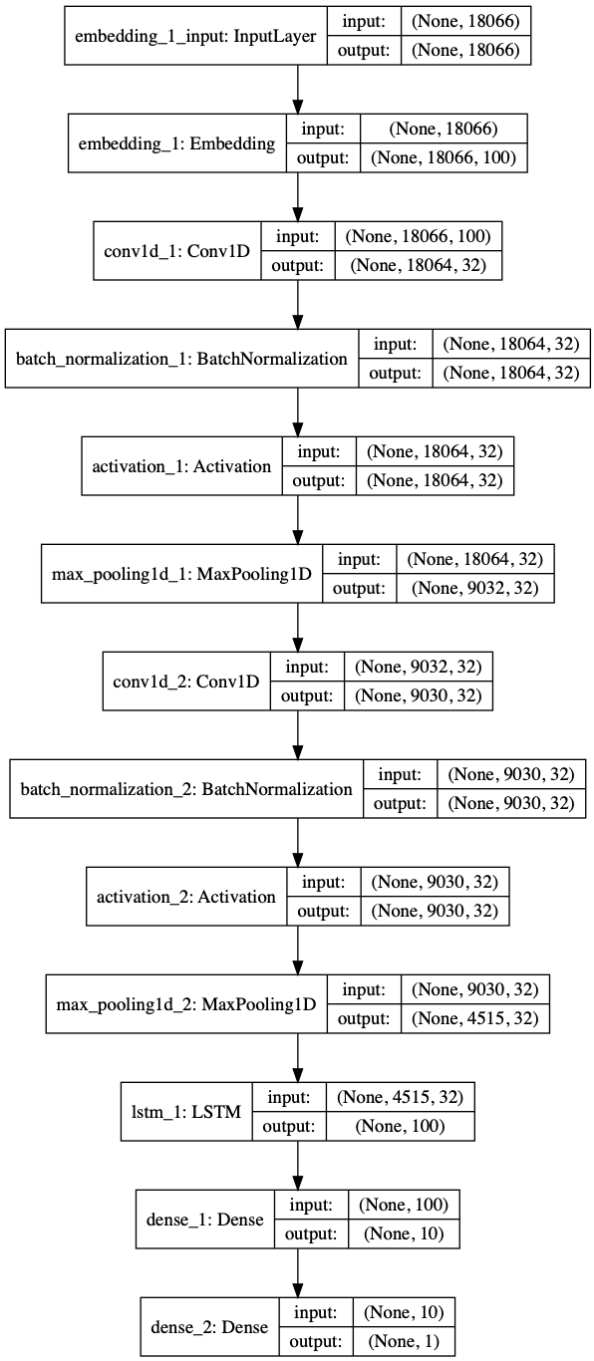


Figure 6.11: CNN-LSTM model (Architecture 2).

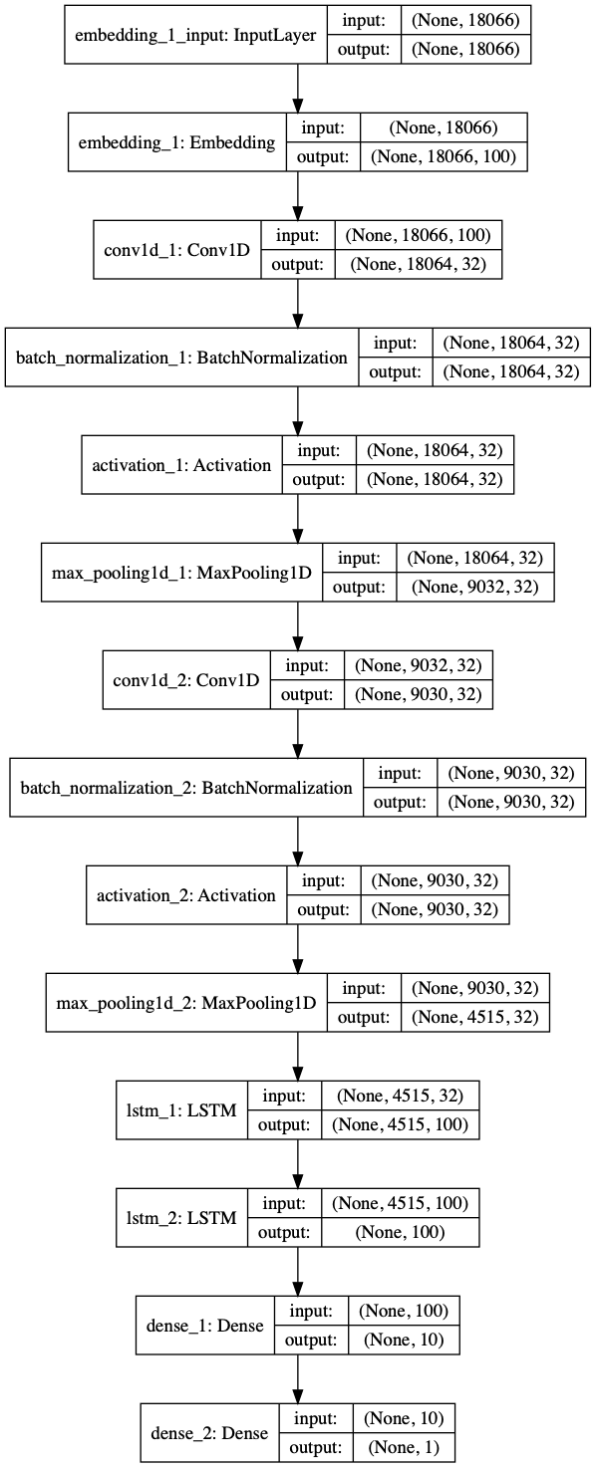


Figure 6.12: CNN-LSTM model (Architecture 3).

6.6.4 CNN-LSTM Architecture Four

We develop the fourth architecture using two CNN layers followed by one LSTM layer. We use 64 filters with the CNN layer, and 100 hidden units with the LSTM layer. The CNN-LSTM layer is followed by two densely connected (DNN) layers with ten, and one hidden units respectively, see Figure 6.13. "relu" activation functions are used with CNN and LSTM layers, while "sigmoid" functions are used with the DNN layers. In this model, we utilized character embeddings to represent the input session sequence. We try to let the model learn these embeddings by considering them as another hyper-parameter that the model needs to learn. The hyper-parameters of this model are: embedding size: 100, CNN filters: 64, filter size: 3, pool size: 2, batch size: 32, epochs: 20, dropout: 0.1, recurrent dropout: 0.1, and L2: 0.01.

A batch normalization is used with LSTM layer to contain a covariate shift by normalizing the activations of the layer. Moreover, L2 norm is used to reduce the learning rate over time.

The Adam optimization method is used to update the model's weights. The optimization process is applied after every 32 samples. The average error over these 32 samples is propagated back using the Adam algorithm.

6.7 Results and Evaluation

We use a 10-fold cross-validation approach to train and test the proposed architectures. Within each fold, the models are tested and validated on different datasets. The testing and validation results from ten folds are concatenated to form one testing and one validation dataset. To evaluate the predicted data, we use ROC curves with AUC scores.

The predicted results are presented using histogram plot. We preview our results in the following order:

1. CNN architecture-based results

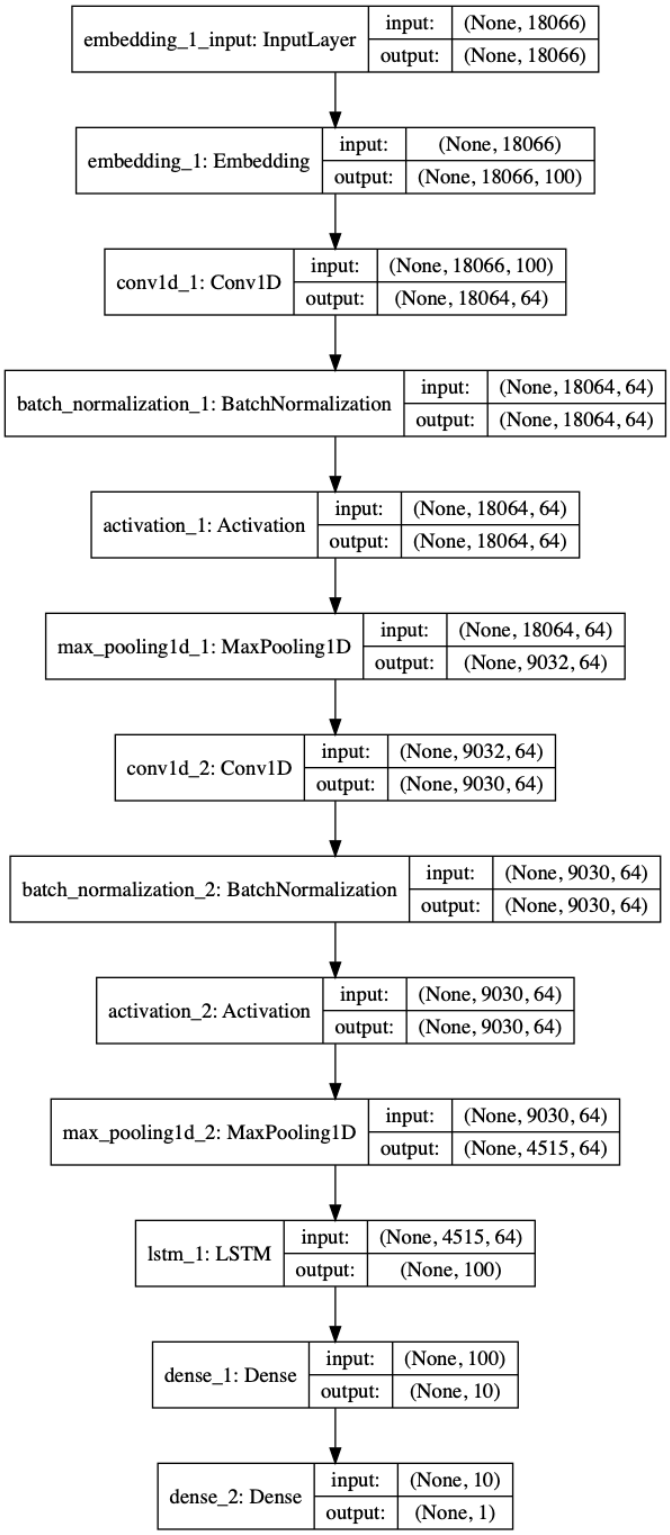


Figure 6.13: CNN-LSTM model (Architecture 4).

2. LSTM architecture-based results

3. CNN-LSTM architecture-based results

The proposed architectures are evaluated using ROC curves. We present our evaluation in the following order:

1. ROCs of CNN-based results

2. ROCs of LSTM-based results

3. ROCs of CNN-LSTM-based results

4. Comparison of total AUC results

CNN ARCHITECTURE-BASED RESULTS

In this section, we present and evaluate the predicted probability scores of using CNN-architecture models. First, we use confusion matrix to show models' performance at 0.5 threshold value. Based on 0.5 threshold value, four CNN-based architectures are evaluated using Precision, Recall, F1 score, and Accuracy metrics. Second, we visualize the results using histogram plots, where the class distributions of both normal and malicious samples are visualized. We use 100 bins to represent the distribution intervals on the histogram. The predicted probability scores are presented with class color. The red color for malicious samples and blue color for normal samples. To optimize the presentation of the histograms, we use the log scale of base 10. The log scale histogram allows high-frequency ranges to be displayed without low-frequency ranges being squeezed down into the bottom of the graph. Third, we used the ROC curves to evaluate the overall performance of the proposed architectures.

6.8 CNN ARCHITECTURE ONE RESULTS

6.8.1 CONFUSION MATRIX

The predicted scores of testing datasets are evaluated using confusion matrix, as shown in Table 6.2. The CNN architecture one model shows a good performance with testing dataset by predicting 47 of the malicious samples correctly and mispredicting 22 samples, or the true positive rate is 68%. On the other side, the model predicts 5006 of the negative samples correctly and mispredicts 2512 samples, or the true negative rate is 67%.

Increasing the CNN model width by considering more hidden units does not improve the model performance compared to the CNN based model proposed in Chapter 5. The false positive rate of CNN architecture one is 33%, while the false positive rate of the base CNN is 26%. On the other side, the CNN architecture one shows slightly low performance of detecting malicious samples compared to the base CNN model. The true positive rate is 68%, while the true positive rate of the base model is 71%, as illustrated in Table 6.5.

Table 6.1: CNN architecture one Confusion Matrix of testing cases. The true positive is 47, and the false positive is 2512. On the other side, the true negative is 5006 and false negative is 22.

N = 7587	**Predicted: Insider**	**Predicted: Normal**
Actual: Insider	47	22
Actual: Normal	2512	5006

6.8.2 RESULTS VISUALIZATION

The predicted probability scores of CNN architecture one model are presented using 100-bar histogram. We use red and blue class colors to represent malicious and normal samples respectively, see Figure 6.14.

We can see that many normal samples are incorrectly predicted with low probabilities through observation of the overlap between intervals. Also, we can see that some malicious

samples are predicted with high probabilities. The left and right sides of Figure 6.14 present the overlap between the normal and malicious samples. The overlap intervals represent the false positive or false negative samples depending on the threshold value.

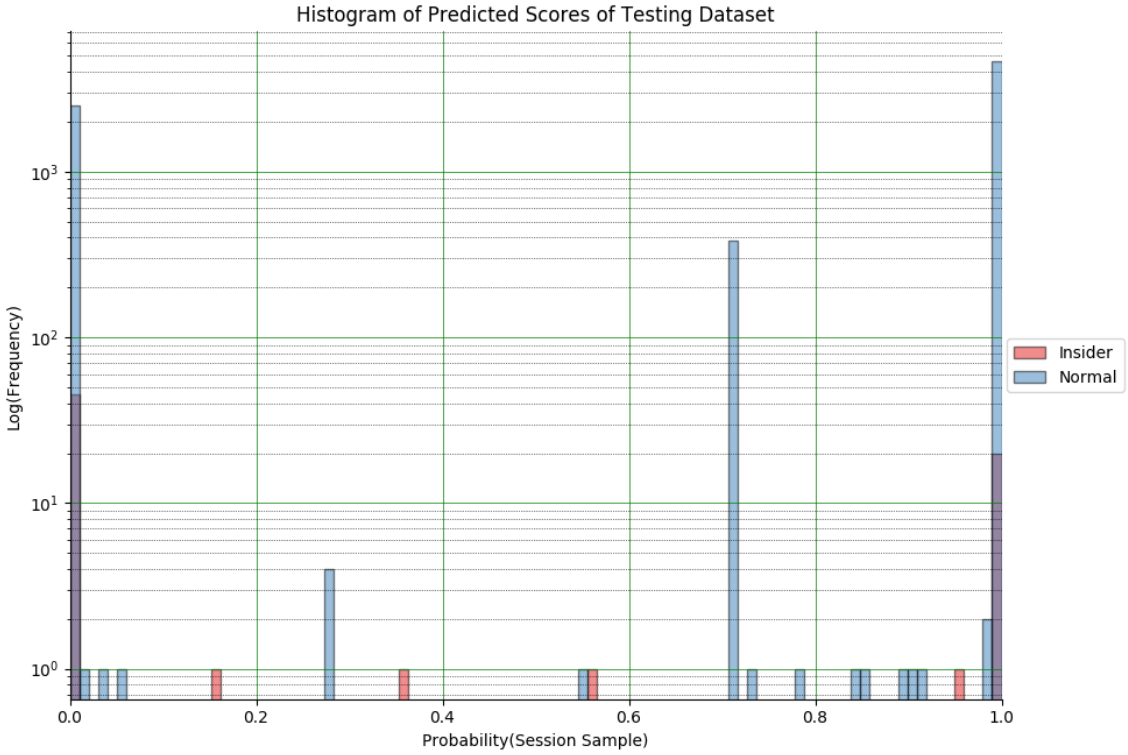


Figure 6.14: Histogram of CNN predicted scores on testing dataset (Architecture One).

6.9 CNN Architecture Two Results

6.9.1 Confusion matrix

The predicted scores of testing datasets are evaluated using confusion matrix, as shown in Table 6.2. The CNN model shows a good performance with testing dataset by predicting 41 of the malicious samples correctly and mispredicting 28 samples, or the true positive rate is 60%. On the other side, the model predicts 6468 of the negative samples correctly and mispredicts 1050 samples, or the true negative rate is 86%.

Increasing model depth using extra CNN layers does improve the model performance of classifying the negative samples compared to the CNN based model proposed in Chapter 5.

The false positive rate of CNN architecture two is 14%, while the false positive rate of the base CNN is 26%. On the other side, the CNN architecture two shows low performance of detecting malicious samples compared to base CNN model. The true positive rate is 60%, while the true positive rate of the base model is 71%, see Table 6.5.

Table 6.2: CNN architecture two Confusion Matrix of testing cases. The true positive is 41, and the false positive is 1050. On the other side, the true negative is 6468 and false negative is 28.

N = 7587	**Predicted: Insider**	**Predicted: Normal**
Actual: Insider	41	28
Actual: Normal	1050	6468

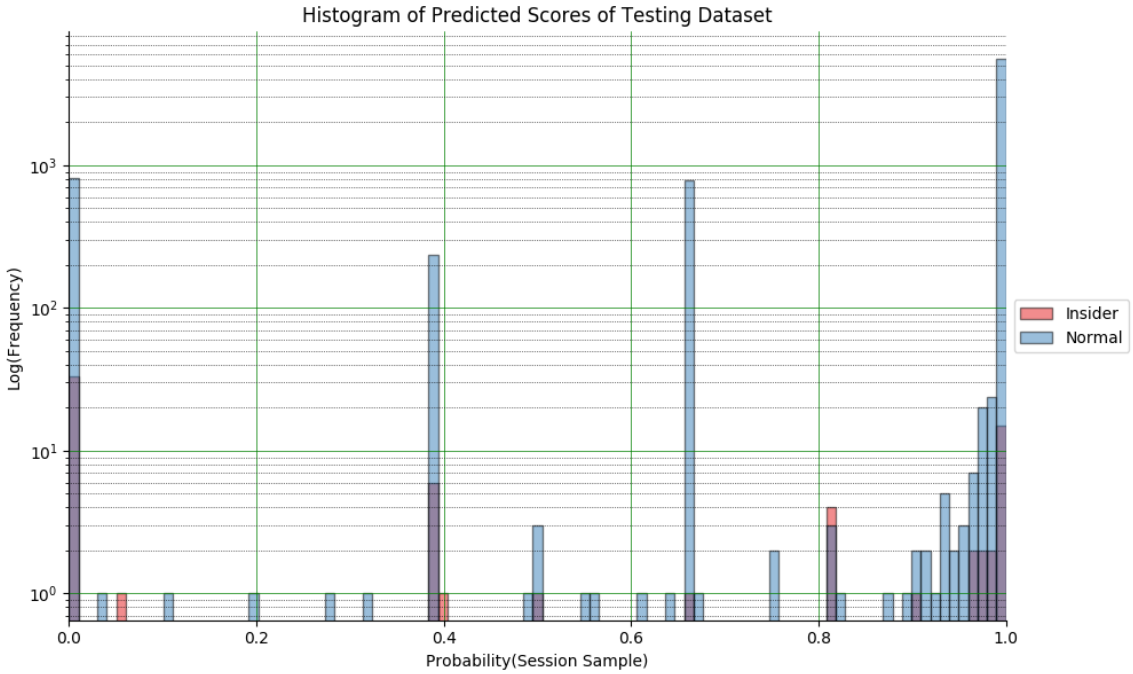


Figure 6.15: Histogram of CNN predicted scores on testing dataset (Architecture Two).

6.9.2 Results Visualization

The predicted probability scores of CNN architecture two model are visualized using a histogram as shown in Figure 6.15.

Through an observation of the overlap between intervals, it becomes clear that many normal samples are falsely predicted with low probabilities. Also, we can see that some malicious samples are predicted with high probabilities. The brown intervals of Figure 6.15 present the overlap between the normal and malicious samples. The overlap intervals represent the false positive or false negative samples depending on the threshold value. Figure 6.15 shows two high blue bars under 0.5 threshold. The bar that close to 0 probability score has log(frequency) close to 10^3, which means its frequency close to 1000. Summing over blue bars under 0.5 threshold value gives 1050, which is the same results that we got in Table 6.2.

6.10 CNN Architecture Three Results

6.10.1 Confusion matrix

The predicted scores of testing datasets are evaluated through the use of a confusion matrix, as illustrated in Table 6.3. The CNN architecture three model shows a good performance with testing dataset by predicting 44 of the malicious samples correctly and mispredicting 25 samples, or the true positive rate is 64%, as depicted in Table 6.5. On the other side, the model predicts 6667 of the negative samples correctly and mispredicts 851 samples, or the true negative rate is 89%.

Increasing model depth using extra CNN layers does improve the model performance of classifying the negative samples compared to the CNN based model proposed in Chapter 5. The false positive rate of CNN architecture three is 11%, while the false positive rate of the base CNN is 26%.

6.10.2 Results Visualization

The predicted probability scores of CNN architecture three model are presented with a histogram, as shown in Figure 6.16.

Table 6.3: CNN architecture three Confusion Matrix of testing cases. The true positive is 44, and the false positive is 851. On the other side, the true negative is 6667 and false negative is 25.

N = 7587	**Predicted: Insider**	**Predicted: Normal**
Actual: Insider	44	25
Actual: Normal	851	6667

The brown intervals of Figure 6.16 represent the overlaps between the normal and malicious samples, where either some of the normal samples are falsely predicted with low probabilities or some of the malicious samples are predicted with high probabilities. The overlap intervals represent the false positive or false negative samples depending on the threshold value.

For instance, we can see that there are several red bars above 0.5 threshold value. This means we have false negative samples. Summing all the red intervals above 0.5 value equals 25 samples, which are the false negative samples as shown in Table 6.3.

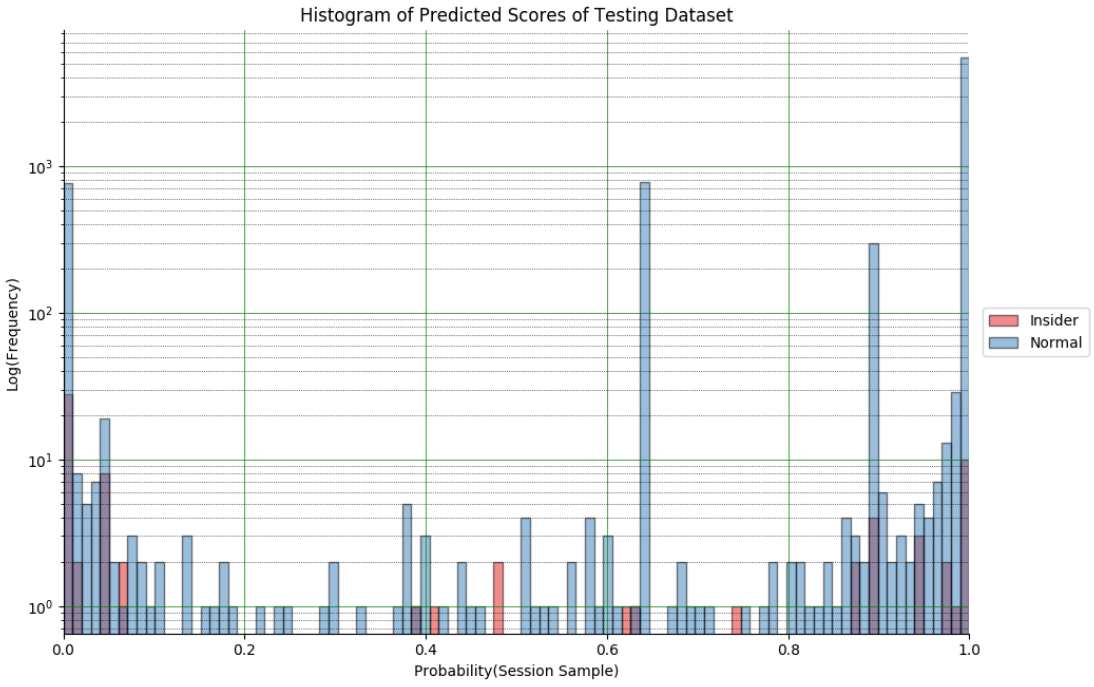


Figure 6.16: Histogram of CNN predicted scores on testing dataset (Architecture Three).

6.11 CNN Architecture Four Results

6.11.1 Confusion matrix

The predicted scores of testing datasets are evaluated using confusion matrix, as shown in Table 6.4. The CNN architecture four model shows a low performance with testing dataset compared to the base model by predicting 44 of the malicious samples correctly and mispredicting 25 samples, or the true positive rate is 64%, see Table 6.5. On the other side, the model predicts 5206 of the negative samples correctly and mispredicts 2312 samples, or the true negative rate is 74%.

Increasing the number of CNN layers does not improve the model performance compared to the base CNN model proposed in Chapter 5. The model shows low performance with both positive and negative samples. The false positive rate of CNN architecture three is 30%, while the false positive rate of the base CNN is 26%.

We use F1 score to show the balance between Precision and Recall, as illustrated in Table 6.5. CNN architecture three obtains the best F1 score.

Table 6.4: CNN architecture four Confusion Matrix of testing cases. The true positive is 44, and the false positive is 2312. On the other side, the true negative is 5206 and false negative is 25.

N = 7587	**Predicted: Insider**	**Predicted: Normal**
Actual: Insider	44	25
Actual: Normal	2312	5206

6.11.2 Results Visualization

The predicted probability scores of CNN architecture four model are presented using a histogram of 100 bars. We differentiate between malicious and normal classes using red and blue colors respectively, as illustrated in Figure 6.17.

We can observe that many normal samples are falsely predicted with low probabilities. Also, we can see that some malicious samples are predicted with high probabilities. The brown intervals of Figure 6.17 represent the overlap between the normal and malicious samples. Moreover, the overlap intervals represent the false positive or false negative samples depending on the threshold value.

For example, there are several blue bars that are located below the 0.5 probability threshold. Summing up over all of those bars results in 2312 false-positive samples.

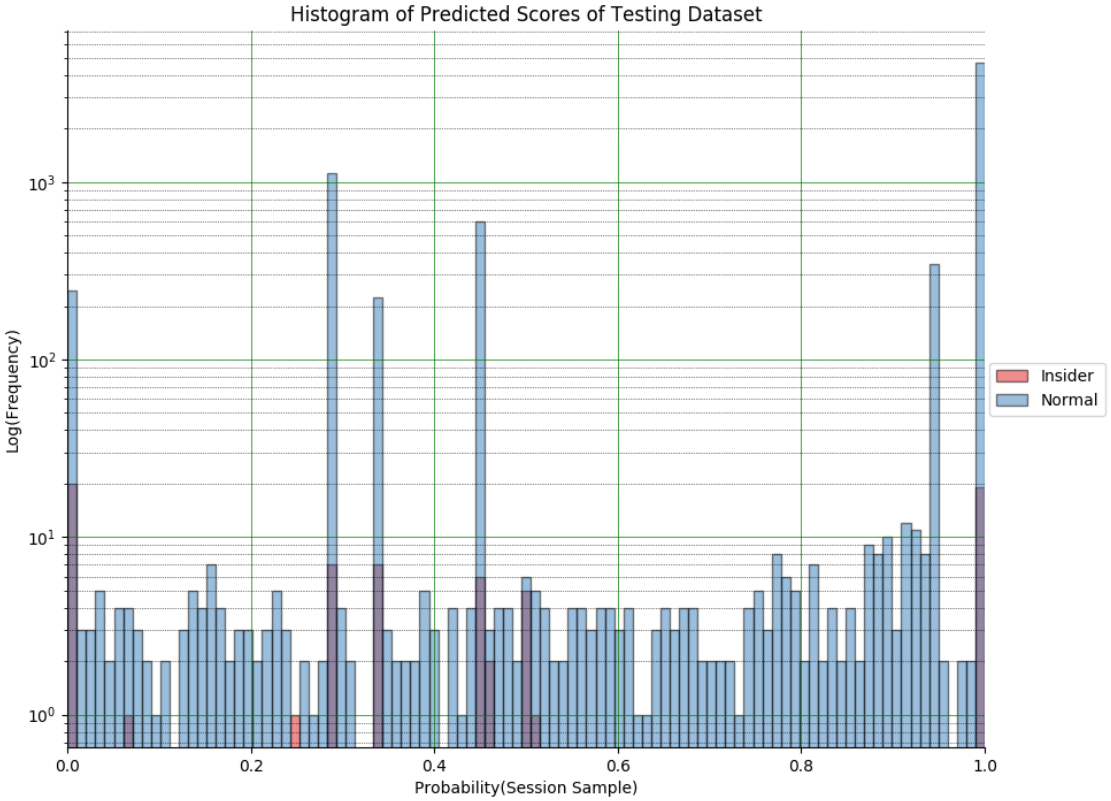


Figure 6.17: Histogram of CNN predicted scores on testing dataset (Architecture Four).

Table 6.5: Performance of CNN-based models. The results of four architectures are presented in term of Precision, Recall, F1, Accuracy, and AUC scores.

CNN-based Model	Precision	Recall	F1 Score	Accuracy	AUC	Epochs
Architecture 1	0.018	0.68	0.036	0.67	0.69	20
Architecture 2	0.038	0.59	0.071	0.86	0.82	20
Architecture 3	0.049	0.64	0.091	0.88	0.85	20
Architecture 4	0.019	0.64	0.036	0.69	0.72	20

In this section, we present and evaluate the predicted probability scores of using LSTM-architecture models. First, we use a confusion matrix to show models' performance at 0.5 threshold value. Second, we visualize the results using histogram plots. We visualize the class distributions of both normal and malicious samples. For this, we use 100 bins to represent the distribution intervals with class color, where the red color is used for malicious samples and the blue color is used for normal samples. To optimize the presentation of the histograms, we use the log scale of base 10. The log scale histogram allows high-frequency ranges to be displayed without low-frequency ranges being squeezed down into the bottom of the graph. Third, we used the ROC curves to evaluate the overall performance of the proposed architectures.

6.12 LSTM ARCHITECTURE ONE RESULTS

6.12.1 CONFUSION MATRIX

The predicted scores of the testing dataset are evaluated using a confusion matrix, as demonstrated in Table 6.6. The LSTM architecture one model illustrates a good performance with the testing dataset through predicting 58 of the malicious samples correctly and mispredicting 11 samples, or in other words the true positive rate is 84%, see Table 6.10. Conversely, the model predicts 5404 of the negative samples correctly and mispredicts 2114 samples; the true negative rate is 72%.

Increasing model width using extra hidden units improves the model ability to classify the normal samples compared to the LSTM base model that is proposed in Chapter 5. However, the model performance with positive samples is lower that of the base model. The LSTM architecture one performs better with the negative samples. The false positive rate of LSTM architecture one is 28%, while the false positive rate of the base LSTM is 37%.

Table 6.6: LSTM architecture one Confusion Matrix of testing cases. The true positive is 58, and the false positive is 2114. On the other side, the true negative is 5404 and false negative is 11.

N = 7587	**Predicted: Insider**	**Predicted: Normal**
Actual: Insider	58	11
Actual: Normal	2114	5404

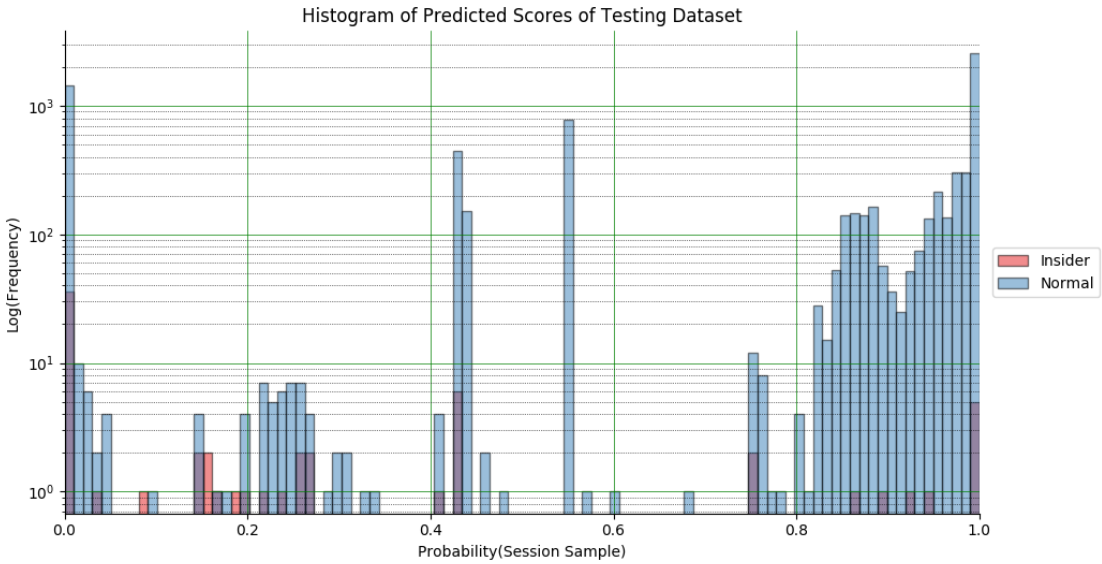


Figure 6.18: Histogram of LSTM predicted scores on testing dataset (Architecture One).

6.12.2 Results Visualization

We visualize the class distribution of the probability scores that are predicted by LSTM architecture one model. As shown in Figure 6.18, the distribution of both normal and malicious samples are depicted with a histogram. 100 bins are used to represent the distribution intervals. Each interval represents a range of probabilities. Figure 6.18 shows these probability intervals with class color, where red and blue colors represent malicious and normal samples, respectively.

By observing the overlap of the intervals, it is evident that many normal samples are falsely predicted with low probabilities. Also, it is evident that some malicious samples are

predicted with high probabilities. The brown intervals of Figure 6.18 represent the overlap of the normal and malicious samples. The overlapping intervals represent the false positive or false negative samples depending on the threshold value.

Based on the above analysis, choosing the threshold value is critical for the model performance. For instance, see Figure 6.18, choosing a threshold value between 0.4 and 0.5 reduces the false-positive samples.

6.13 LSTM Architecture Two Results

6.13.1 Confusion matrix

The predicted scores of the testing dataset are evaluated using confusion matrix, as shown in Table 6.7. The LSTM architecture two model shows a good performance with testing dataset by predicting 55 of the malicious samples correctly and mispredicting 14 samples, or the true positive rate is 80%, as illustrated in Table 6.10. On the other side, the model predicts 5324 of the negative samples correctly and mispredicts 2194 samples , or the true negative rate is 71%.

Increasing model depth using extra LSTM layers improves the model ability to classify the normal samples compared to the LSTM base model proposed in Chapter 5. The LSTM architecture two performs better with the negative samples. The false-positive rate of LSTM architecture two is 29%, while the false-positive rate of the base LSTM is 37%. However, the model shows a low performance with positive samples.

Table 6.7: LSTM architecture two Confusion Matrix of testing cases. The true positive is 55, and the false positive is 2194. On the other side, the true negative is 5324 and false negative is 14.

N = 7587	**Predicted: Insider**	**Predicted: Normal**
Actual: Insider	55	14
Actual: Normal	2194	5324

6.13.2 Results Visualization

The predicted probability scores of LSTM architecture two model are visualized using a 100-bar histogram with class color, as shown in Figure 6.19.

By observing the overlap intervals, we can see that many normal samples are falsely predicted with low probabilities. Also, we can see that some malicious samples are predicted with high probabilities. The brown intervals of Figure 6.19 present the overlap between the normal and malicious samples. The overlap intervals represent the false positive or false negative samples depending on the threshold value.

We can observe that the false positive is high. There are two blue bars close to the log of 10^3 lead to high false-positive samples. Also, we can observe that there are several overlap red bars above 0.5 threshold value lead to false-negative samples.

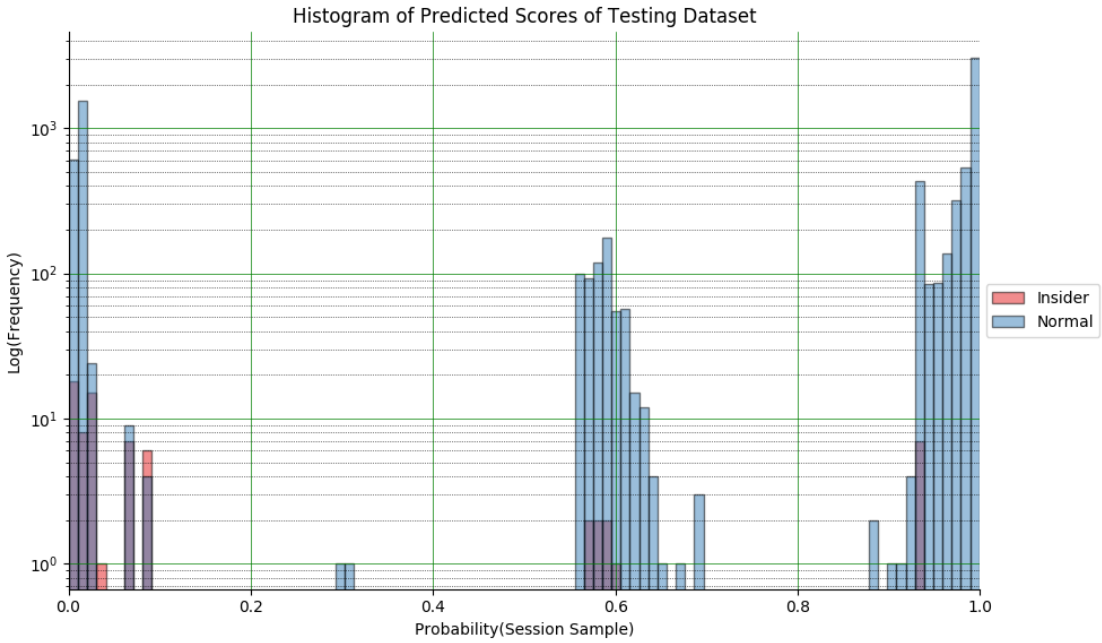


Figure 6.19: Histogram of LSTM predicted scores on testing dataset (Architecture Two).

6.14 LSTM Architecture Three Results

6.14.1 Confusion matrix

The predicted scores of testing datasets are evaluated using confusion matrix, as shown in Table 6.8. The LSTM architecture three model shows a good performance with testing dataset by predicting 62 of the malicious samples correctly and mispredicting 7 samples, or the true positive rate is 90%, as shown in Table 6.10. On the other side, the model predicts 5490 of the negative samples correctly and mispredicts 2028 samples, or the true negative rate is 73%.

Increasing model depth using extra LSTM layers improves the model performance compared to the LSTM base model proposed in Chapter 5. The model performance with positive samples are very close to the base model. However, the LSTM architecture three performs better with the negative samples. The false positive rate of LSTM architecture three is 27%, while the false positive rate of the base LSTM is 37%.

Table 6.8: LSTM architecture three Confusion Matrix of testing cases. The true positive is 62, and the false positive is 2028. On the other side, the true negative is 5490 and false negative is 7.

N = 7587	**Predicted: Insider**	**Predicted: Normal**
Actual: Insider	62	7
Actual: Normal	2028	5490

6.14.2 Results Visualization

The predicted probability scores of LSTM architecture three model are presented using a histogram of 100 bars. Class colors are used to differentiate between normal and malicious samples, as illustrated in Figure 6.20.

The brown intervals of Figure 6.20 represent the overlaps between normal and malicious samples. The overlap intervals represent the false positive or false negative samples

depending on the threshold value.

We can estimate that there are high false-positive samples by observing the high blue bars on the left side of Figure 6.20 under 0.5 threshold value.

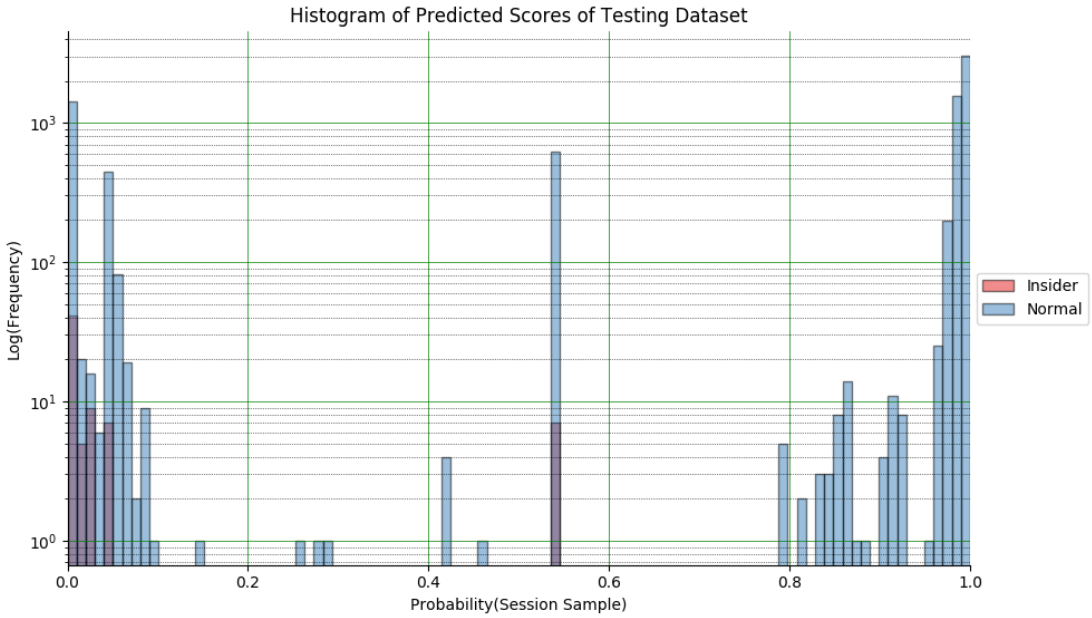


Figure 6.20: Histogram of LSTM predicted scores on testing dataset (Architecture Three).

6.15 LSTM Architecture Four Results

6.15.1 Confusion matrix

The predicted scores of testing datasets are evaluated using confusion matrix, as shown in Table 6.9. The LSTM architecture four model shows a better performance with testing dataset compared to the base model by predicting 60 of the malicious samples correctly and mispredicting 9 samples, or the true positive rate is 87%, see Table 6.10. On the other side, the model predicts 6108 of the negative samples correctly and mispredicts 1410 samples, or the true negative rate is 81%.

Increasing the number of hidden units in the LSTM layer and increasing the number of LSTM layers improve the model performance compared to the LSTM base model proposed in Chapter 5. The model performance with positive samples are very similar to the base

model. However, the LSTM architecture four performs better with the negative samples. The false positive rate of LSTM architecture four is 19%, while the false positive rate of the base LSTM is 37%.

Table 6.9: LSTM architecture four Confusion Matrix of testing cases. The true positive is 60, and the false positive is 1410. On the other side, the true negative is 6108 and false negative is 9.

N = 7587	**Predicted: Insider**	**Predicted: Normal**
Actual: Insider	60	9
Actual: Normal	1410	6108

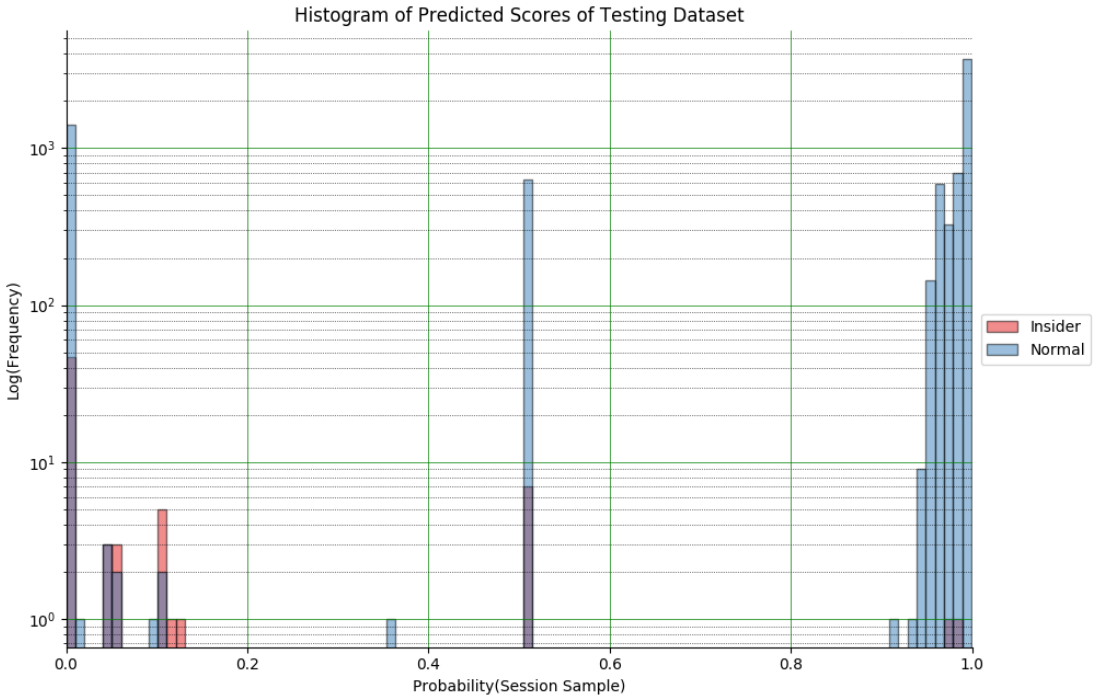


Figure 6.21: Histogram of LSTM predicted scores on testing dataset (Architecture Four).

6.15.2 Results Visualization

We visualize the class distribution of both normal and malicious samples using a 100-bin histogram, as shown in Figure 6.21.

The brown intervals of Figure 6.21 represent the overlaps between normal and malicious samples. By observing the overlap intervals, we can see that some of the normal samples are falsely predicted with low probabilities, and some of the malicious samples are predicted with high probabilities. The overlap intervals represent the false positive or false negative samples depending on the threshold value.

We can see that the overlap blue bars below 0.5 threshold result in 1410 false positive samples as shown in Table 6.9. On the other side, the few red bars above 0.5 threshold value lead to 9 false-negative samples.

The insider dataset shows unbalanced class distributions. Thus, we use F1 score that represents the weighted average of Precision and Recall, see Table 6.10. LSTM architecture four achieves the best F1 score.

Table 6.10: Performance of LSTM-based models. The results of four architectures are presented in term of Precision, Recall, F1, Accuracy, and AUC scores.

LSTM-based Model	Precision	Recall	F1 Score	Accuracy	AUC	Epochs
Architecture 1	0.028	0.84	0.052	0.72	0.8	20
Architecture 2	0.024	0.8	0.047	0.71	0.77	20
Architecture 3	0.03	0.9	0.057	0.73	0.84	20
Architecture 4	0.04	0.87	0.078	0.81	0.86	20

CNN-LSTM Architecture-based Results

In this section, we present and evaluate the predicted probability scores of using CNN-LSTM-architecture models. First, we use confusion matrix to show models' performance at 0.5 threshold value. Second, we visualize the results using histogram plots. We visualize the class distributions of both normal and malicious samples. For this, we use 100 bins to represent the distribution intervals with class color, where the red color for malicious samples and the blue color for normal samples. To optimize the presentation of the histograms, we use the log scale of base 10. The log scale histogram allows high-frequency ranges to be displayed without low-frequency ranges being squeezed down into the bottom of the

graph. Third, we used the ROC curves to evaluate the overall performance of the proposed architectures.

6.16 CNN-LSTM Architecture One Results

6.16.1 Confusion matrix

The predicted scores of testing datasets are evaluated using confusion matrix, as shown in Table 6.11. The CNN-LSTM architecture one model shows a low performance with testing dataset compared to the base model by predicting 40 of the malicious samples correctly and mispredicting 29 samples, or the true positive rate is 58%, see Table 6.15. On the other side, the model predicts 4710 of the negative samples correctly and mispredicts 2808 samples, or the true negative rate is 63%.

Increasing the number of CNN-LSTM layers does not improve the model performance compared to the base CNN-LSTM model proposed in Chapter 5. The model shows low performance with both positive and negative samples. The false positive rate of CNN-LSTM architecture one is 37%, while the false positive rate of the base CNN-LSTM is 17%.

Table 6.11: CNN-LSTM architecture one Confusion Matrix of testing cases. The true positive is 40, and the false positive is 2808. On the other side, the true negative is 4710 and false negative is 29.

N = 7587	**Predicted: Insider**	**Predicted: Normal**
Actual: Insider	40	29
Actual: Normal	2808	4710

6.16.2 Results Visualization

The predicted probability scores of CNN-LSTM architecture one model are visualized using a 100-bin histogram, as shown in Figure 6.22. We use class colors to differentiate

between the distributions of normal and malicious samples, where red and blue colors are used for malicious and normal samples respectively.

The brown intervals of Figure 6.22 present the overlap between normal and malicious samples. The overlap intervals represent the false positive or false negative samples depending on the threshold value. By observing the overlap intervals, we can see that some normal samples are falsely predicted with low probabilities, and some malicious samples are falsely predicted with high probabilities.

We can see that there is one instance of high overlap bars below 0.5 threshold value leads to high false positive samples, see Figure 6.22. On the other side, we can see that there are several overlap bars above 0.5 represent 29 false-negative samples, see Table 6.11.

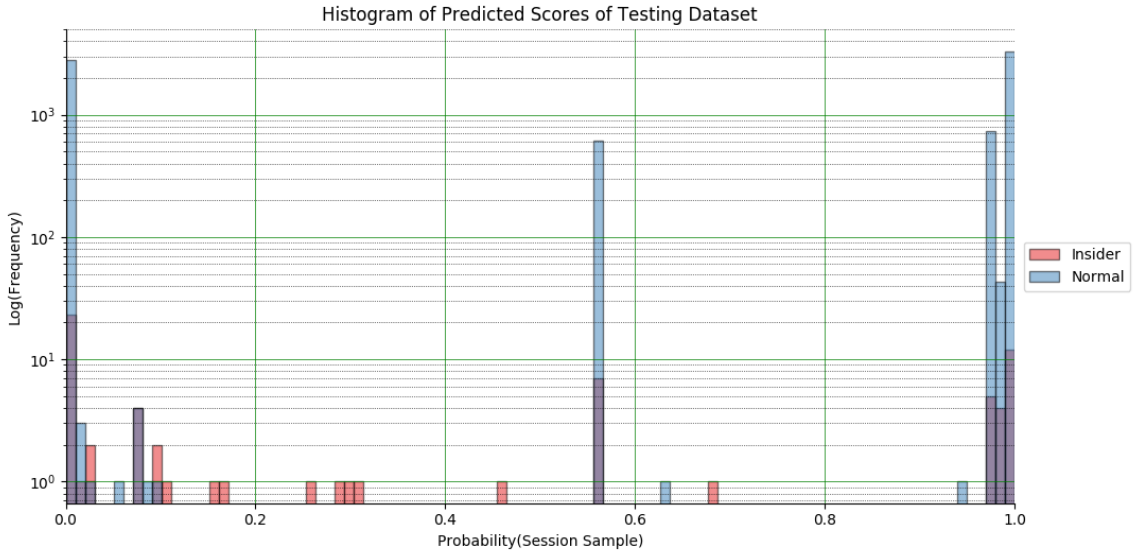


Figure 6.22: Histogram of CNN - LSTM predicted scores on testing dataset (Architecture One).

6.17 CNN-LSTM Architecture Two Results

6.17.1 Confusion matrix

The predicted scores of testing datasets are evaluated using confusion matrix, as shown in Table 6.12. The CNN-LSTM architecture two model shows a better performance with

positive samples compared to the base model by predicting 67 of the malicious samples correctly and mispredicting 2 samples, or the true positive rate is 97%, as shown in Table 6.15. On the other side, the model predicts 4718 of the negative samples correctly and mispredicts 2800 samples, or the true negative rate is 63%.

Increasing the number of CNN-LSTM layers does improve the model ability to detect the malicious samples compared to the base CNN-LSTM model proposed in Chapter 5. However, the model shows low performance with the negative samples. The false positive rate of CNN-LSTM architecture two is 37%, while the false positive rate of the base CNN-LSTM is 17%.

Table 6.12: CNN-LSTM architecture two Confusion Matrix of testing cases. The true positive is 67, and the false positive is 2800. On the other side, the true negative is 4718 and false negative is 2.

N = 7587	**Predicted: Insider**	**Predicted: Normal**
Actual: Insider	67	2
Actual: Normal	2800	4718

6.17.2 Results Visualization

The predicted probability scores of CNN-LSTM architecture two model are presented using a 100-bar histogram. To visualize both malicious and normal probability scores, we use class color, where red and blue colors used for malicious and normal samples respectively.

The brown intervals of Figure 6.23 present the overlap between normal and malicious samples. By observing Figure 6.23, we can see that some normal samples are falsely predicted with low probabilities, and some malicious samples are predicted with high probabilities. The overlap intervals represent the false positive or false negative samples depending on the threshold value.

We can see that there is one instance of high blue overlapping bars that leads to high false positive samples, see Figure 6.23 close to zero.

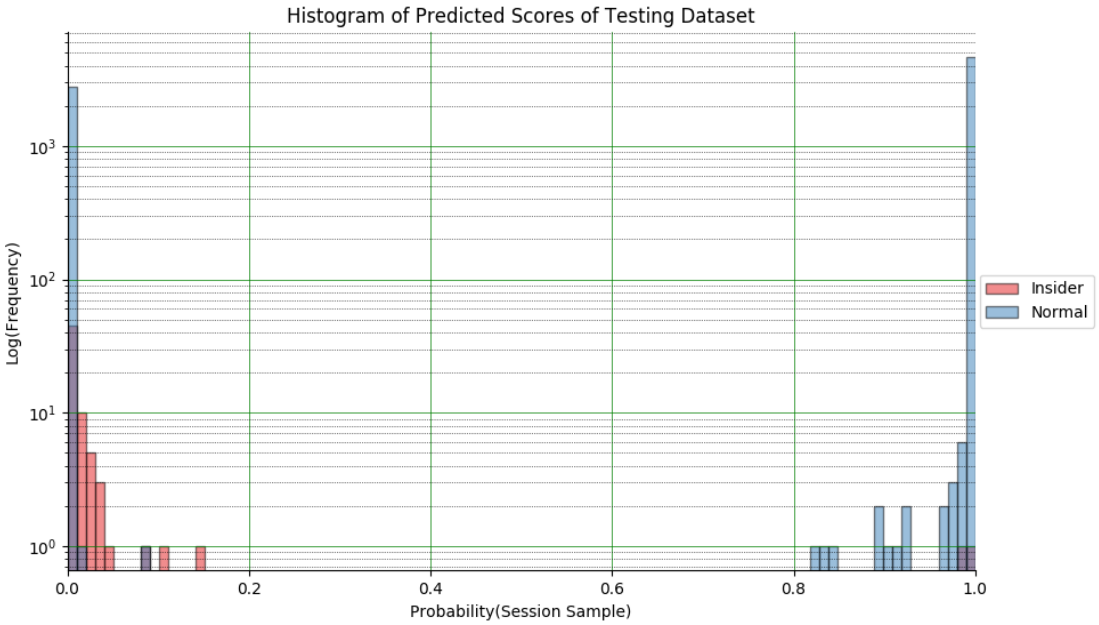


Figure 6.23: Histogram of CNN - LSTM predicted scores on testing dataset (Architecture Two).

6.18 CNN-LSTM Architecture Three Results

6.18.1 Confusion matrix

The predicted scores of testing datasets are evaluated using confusion matrix, as shown in Table 6.13. The CNN-LSTM architecture three model shows a low performance with testing dataset compared to the base model by predicting 48 of the malicious samples correctly and mispredicting 21 samples, or the true positive rate is 70%, as illustrated in Table 6.15. On the other side, the model predicts 4700 of the negative samples correctly and mispredicts 2818 samples, or the true negative rate is 63%.

Increasing the number of CNN-LSTM layers does not improve the model performance compared to the base CNN-LSTM model proposed in Chapter 5. The model shows low performance with both positive and negative samples. The false positive rate of CNN-LSTM architecture three is 37%, while the false positive rate of the base CNN-LSTM is 17%.

Table 6.13: CNN-LSTM architecture three Confusion Matrix of testing cases. The true positive is 48, and the false positive is 2818. On the other side, the true negative is 4700 and false negative is 21.

N = 7587	**Predicted: Insider**	**Predicted: Normal**
Actual: Insider	48	21
Actual: Normal	2818	4700

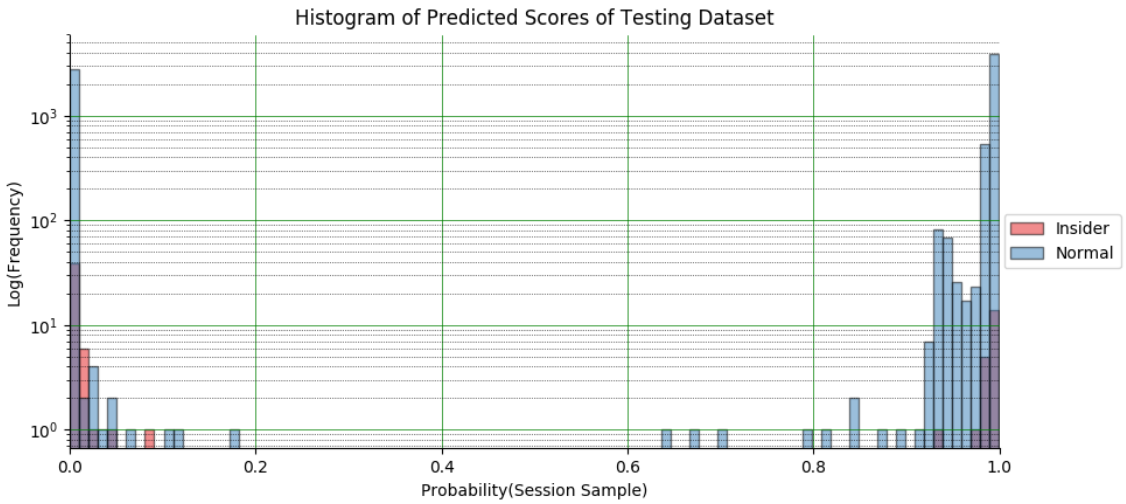


Figure 6.24: Histogram of CNN - LSTM predicted scores on testing dataset (Architecture Three).

6.18.2 Results Visualization

We visualize the class distribution of both normal and malicious samples using a histogram, as shown in Figure 6.24. We use 100 bins to represent the distribution intervals on the histogram. Also, the predicted probability scores of CNN-LSTM architecture three model are presented with class colors to differentiate between classes.

The brown intervals of Figure 6.24 present the overlap between normal and malicious samples. The overlap intervals represent the false positive or false negative samples depending on the threshold value.

Figure 6.24 shows one instance of high blue overlap bars below 0.5 threshold value, which leads to high false positive samples, see Figure 6.24. On the other side, above 0.5 value, there are four overlap bars represent 21 false-negative samples, see Table 6.13.

6.19 CNN-LSTM Architecture Four Results

6.19.1 Confusion matrix

The predicted scores of testing datasets are evaluated using confusion matrix, as shown in Table 6.14. The CNN-LSTM architecture four model shows a low performance with testing dataset compared to the base model by predicting 41 of the malicious samples correctly and mispredicting 28 samples, or the true positive rate is 60%, see Table 6.15. On the other side, the model predicts 4717 of the negative samples correctly and mispredicts 2801 samples, or the true negative rate is 63%.

Increasing the number of CNN-LSTM layers does not improve the model performance compared to the base CNN-LSTM model proposed in Chapter 5. The model shows low performance with both positive and negative samples. The false positive rate of CNN-LSTM architecture four is 37%, while the false positive rate of the base CNN-LSTM is 17%.

We have a dataset with unbalanced class distributions. Thus, we use F1 score to find the balance between Precision and Recall, as shown in Table 6.15. CNN-LSTM architecture two obtains the best F1 score.

Table 6.14: CNN-LSTM architecture four Confusion Matrix of testing cases. The true positive is 41, and the false positive is 2801. On the other side, the true negative is 4717 and false negative is 28.

N = 7587	**Predicted: Insider**	**Predicted: Normal**
Actual: Insider	41	28
Actual: Normal	2801	4717

6.19.2 Results Visualization

The predicted probability scores of CNN-LSTM architecture four model are presented with 100-bar histogram. The 100 bars represent distribution intervals of both malicious and normal samples. To distinguish between malicious and normal samples, we use class colors, where red and blue colors are used for malicious and normal samples respectively.

Figure 6.25 presents overlaps between normal and malicious samples above and below 0.5 threshold value. The overlap intervals represent the false positive or false negative samples depending on the threshold value.

For example, there is one instance of high overlap bars below 0.5 leads to high false-positive samples. Summing over all blue bars below 0.5 results in 2801 false positive samples, as illustrated Table 6.14. On the other side, there are several red bars result in 28 false-negative samples.

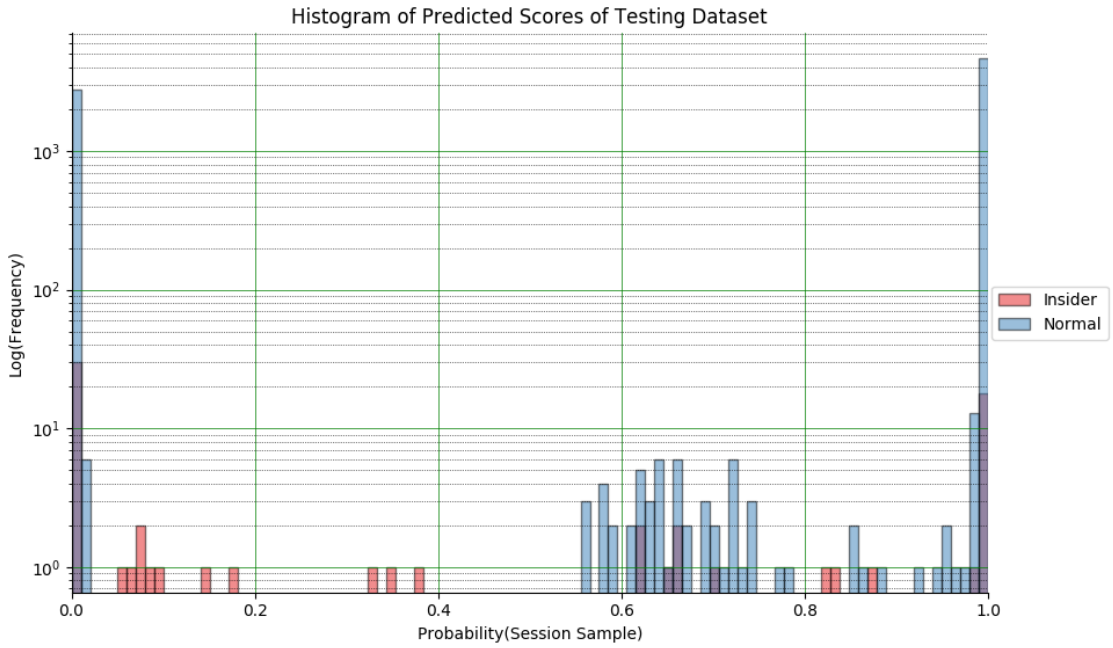
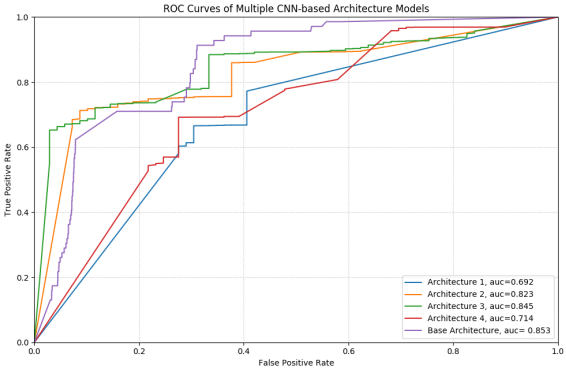


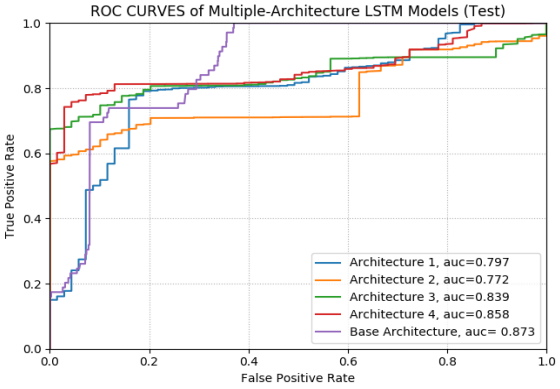
Figure 6.25: Histogram of CNN - LSTM predicted scores on testing dataset (Architecture Four).

Table 6.15: Performance of CNN-LSTM-based models. The results of four architectures are presented in term of Precision, Recall, F1, Accuracy, and AUC scores.

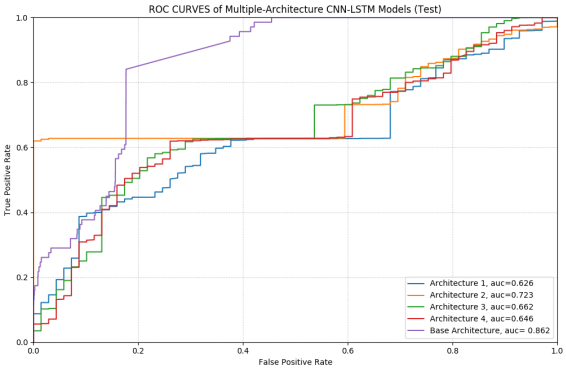
CNN-LSTM Model	Precision	Recall	F1 Score	Accuracy	AUC	Epochs
Architecture 1	0.014	0.6	0.027	0.63	0.63	20
Architecture 2	0.023	0.97	0.045	0.63	0.72	20
Architecture 3	0.017	0.7	0.033	0.63	0.66	20
Architecture 4	0.014	0.6	0.028	0.63	0.65	20



(a) AUC of CNN Architecture-based models.



(b) AUC of LSTM Architecture-based models.



(c) AUC of CNN-LSTM Architecture-based models.

Figure 6.26: ROC curves of CNN, LSTM, and CNN-LSTM models.

6.20 Visual-based Evaluation (ROC)

In this section, we present our visual evaluation of the proposed deep learning-based architectures: CNN-, LSTM-, and CNN-LSTM-based architectures.

For each set of models that includes four models, we draw the ROC curves on one

canvas. Also, we include the ROC curve of the base model that belongs to that set. For example, we draw the four CNN-architecture based models on one canvas in addition to the base CNN model from chapter 5, as shown in Figure 6.26 a

Figure 6.26 shows separate five ROC curves for each of the CNN-, LSTM-, and CNN-LSTM-based architectures. Each plot has the four architectures that are proposed in chapter 6, in addition to the base architecture proposed in Chapter 5.

CHAPTER 7

CONCLUSION AND FUTURE WORK

7.1 CHAPTER OVERVIEW

Insider threat is a growing problem in many organizations. Detecting insider misuse requires an understanding of the behavior of each user. However, identifying unusual user behaviors is a challenging task. Current approaches include profiling, predefined signatures, and machine learning. In this work, we show that insider misuse can be pointed out by developing different structure machine learning models that are learned on session-based data samples. In summary, we notice that research into identifying insider's misuse is important and is a challenging problem.

The number of malicious insiders examples is minimal when compared to normal examples. This leads to imbalanced data sets. The difference in the distribution of the imbalanced dataset makes the learning of supervised algorithms challenging. Moreover, the potential to use natural language processing to extract the right features that can be linked with insider threat hasn't been studied sufficiently.

7.2 RESEARCH SUMMARY

To address the above limitations, we set up two approaches to detect malicious insider threats based on computer-based activities: a probabilistic graphical model-based approach and a deep learning-based approach.

7.2.1 A Probabilistic Graphical Model-based Approach

For our probabilistic graphical model-based approach, we propose an unsupervised model for insider's misuse detection. That is, we develop Stochastic Gradient Descent method to learn Hidden Markov Models (SGD-HMM) with the goal of analyzing user log data. We propose the use of varying granularity levels to represent users' log data: Session-based, Day-based, and Week-based. A user's normal behavior is modeled using SGD-HMM. The model is used to detect any deviation from the normal behavior. We also propose a Sliding Window Technique (SWT) to identify malicious activity by considering the near history of the user's activities. We evaluate the experimental results in terms of Receiver Operating Characteristic (ROC). The area under the curve (AUC) represents the model's performance with respect to the separability of the classes. The higher the AUC scores, the better the model's performance. Combining SGD-HMM with SWT resulted in AUC values between 0.81 and 0.9 based on the window size. Our solution is superior to current solutions based on the achieved AUC scores.

7.2.2 Deep Learning-based Approach

For our deep learning-based approach, we propose a supervised model for insider's misuse detection. We present our solution using natural language processing with deep learning. We examine textual event logs to investigate the semantic meaning behind a user's behavior. The proposed approaches consist of character embeddings and deep learning networks that involve Convolutional Neural Network (CNN) and Long Short-Term Memory (LSTM). We develop three deep-learning models: CNN, LSTM, and CNN-LSTM. We run a 10-fold subject-independent cross-validation procedure to evaluate the developed models. Our deep learning-based approach shows promising behavior. The first base model, CNN, presents a good performance of classifying normal samples with an AUC score of 0.85, false-negative rate 29%, and 26% false-positive rate. The second model, LSTM, shows the best performance of detecting malicious samples with an AUC score of 0.873, false-

negative rate 0%, and 37% false-positive rate. The third model, CNN-LSTM, presents a moderate behavior of detecting both normal and insider samples with an AUC score of 0.862, false-negative rate 16%, and 17% false-positive rate.

7.2.3 Exploration Study

In our work of chapter 6, we investigate 12 different architectures of the deep-base models. We use our proposed deep-based approaches in chapter 5 to investigate networks with deeper and wider structures. For this, we study the impact of increasing the number of CNN or LSTM layers, nodes per layer, and both of them at the same time on the model performance. As a result, we found that increasing the number of hidden units or increasing the depth of the deep learning-based models degrades the performance of the CNN-LSTM models. However, the new second and third CNN architecture models decrease the false positive rate. With the LSTM-based models, the new architectures reduce the false positive rates and increase the false negative rates.

We believe that changing the position of the Batch normalization layer in the proposed deep-based architecture affects models' performance. For instance, set the batch normalization layer after the input layer, or after or before the activation function of the CNN layer could lead to different model behaviors. For this, we will investigate the effect of using CNN layer with Batch normalization layer in our future work.

7.3 Future Work

The future work of this dissertation is as follows:

We aim to develop the deep learning-based approach by:

1. Using word-based embedding layer instead of letter-based. Word embedding presents a set of features for each word based on the embedding dimension. A well-trained word embedding will group the words that represent the normal behaviors apart from the words that represent the malicious behaviors. Thus, the words semantic mean-

ing could be close to or apart from each other based on word-feature distributions. Moreover, considering a sequence of words could give more insights about the semantic meaning of user behaviors. For this, we aim to study the effect of training our proposed models using word embedding.

2. Using bidirectional sequence LSTM to evaluate the semantic meaning of the textual samples. Bidirectional LSTM processes the input textual sequence from a positive and negative time direction. By using the bidirectional LSTM, we can get extra meaning to the network, which may result in further enhancement of the ability of learning text representation.

3. Using Multiple Channel Convolutional Neural Networks (MCCNN). MCCNN processes the textual input samples from different perspectives using different kernal sizes. This would help the model to learn different relations from the input data.

Moreover, we aim to develop our Hidden Markov Model approach by using mixture HMM. This can be done by specifying a separate symbol matrix for each domain.

BIBLIOGRAPHY

[1] *Verizon: 2019 data breach investigations report*, Computer Fraud Security **2019** (2019), no. 6, 4.

[2] J. Johnson A. Karpathy and F.-F. Li., *Course notes on cs231n: Convolutional neural networks for visual recognition*, `https://cs231n.github.io`, 2019.

[3] AM Abirami and V Gayathri, *A survey on sentiment analysis methods and approach*, 2016 Eighth International Conference on Advanced Computing (ICoAC), IEEE, 2017, pp. 72–76.

[4] Z. Alibadi and J.. Vidal, *To read or to do? that's the task*, In Proceedings of the 2018 International Conference on Data Science (ICDATA'18) (2018), 279–285.

[5] Frederico Araujo, Kevin W Hamlen, Sebastian Biedermann, and Stefan Katzenbeisser, *From patches to honey-patches: Lightweight attacker misdirection, deception, and disinformation*, Proceedings of the 2014 ACM SIGSAC conference on computer and communications security, ACM, 2014, pp. 942–953.

[6] Amos Azaria, Ariella Richardson, Sarit Kraus, and VS Subrahmanian, *Behavioral analysis of insider threat: A survey and bootstrapped prediction in imbalanced data*, IEEE Transactions on Computational Social Systems **1** (2014), no. 2, 135–155.

[7] Giampaolo Bella, Stefano Bistarelli, Simon N Foley, and Barry O'Sullivan, *Applications of constraint satisfaction and programming to computer security problems*, (2005).

[8] Yoshua Bengio, Patrice Simard, and Paolo Frasconi, *Learning long-term dependencies with gradient descent is difficult*, IEEE transactions on neural networks **5** (1994), no. 2, 157–166.

[9] Taweh Beysolow II, *Applied Natural Language Processing with Python*, 2018.

[10] Kaushal Bhavsar and Bhushan Trivedi, *Predicting insider threats by behavioural analysis using deep learning*, Proceedings of the International Conference on Secu-

rity and Management (SAM), The Steering Committee of The World Congress in Computer Science, 2018, pp. 97–101.

[11] Jeff A Bilmes et al., *A gentle tutorial of the em algorithm and its application to parameter estimation for gaussian mixture and hidden markov models*, International Computer Science Institute **4** (1998), no. 510, 126.

[12] Brock Böse, Bhargav Avasarala, Srikanta Tirthapura, Yung-Yu Chung, and Donald Steiner, *Detecting insider threats using radish: A system for real-time anomaly detection in heterogeneous data streams*, IEEE Systems Journal **11** (2017), no. 2, 471–482.

[13] Brian M Bowen, Shlomo Hershkop, Angelos D Keromytis, and Salvatore J Stolfo, *Baiting inside attackers using decoy documents*, International Conference on Security and Privacy in Communication Systems, Springer, 2009, pp. 51–70.

[14] Dawn M Cappelli, Andrew P Moore, and Randall F Trzeciak, *The cert guide to insider threats: how to prevent, detect, and respond to information technology crimes (theft, sabotage, fraud)*, Addison-Wesley, 2012.

[15] Varun Chandola, Arindam Banerjee, and Vipin Kumar, *Anomaly detection: A survey*, ACM computing surveys (CSUR) **41** (2009), no. 3, 15.

[16] Nitesh V. Chawla, Kevin W. Bowyer, Lawrence O. Hall, and W. Philip Kegelmeyer, *Smote: synthetic minority over-sampling technique*, Journal of artificial intelligence research **16** (2002), 321–357.

[17] You Chen and Bradley Malin, *Detection of anomalous insiders in collaborative environments via relational analysis of access logs*, Proceedings of the first ACM conference on Data and application security and privacy, ACM, 2011, pp. 63–74.

[18] Clearswift, *insider threat 74 security incidents come extended enterprise nothacking groups*, May 2018.

[19] Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa, *Natural language processing (almost) from scratch*, Journal of machine learning research **12** (2011), no. Aug, 2493–2537.

[20] Carl Colwill, *Human factors in information security: The insider threat–who can you trust these days?*, Information security technical report **14** (2009), no. 4, 186–196.

[21] Daniel L Costa, Matthew L Collins, Samuel J Perl, Michael J Albrethsen, George J Silowash, and Derrick L Spooner, *An ontology for insider threat indicators development and applications*, Tech. report, CARNEGIE-MELLON UNIV PITTSBURGH PA SOFTWARE ENGINEERING INST, 2014.

[22] Brian D Davison and Haym Hirsh, *Predicting sequences of user actions*, Notes of the AAAI/ICML 1998 Workshop on Predicting the Future: AI Approaches to Time-Series Analysis, 1998, pp. 5–12.

[23] Derui Ding, Qing-Long Han, Yang Xiang, Xiaohua Ge, and Xian-Ming Zhang, *A survey on security control and attack detection for industrial cyber-physical systems*, Neurocomputing **275** (2018), 1674–1683.

[24] CERT Division and ExactData LLC, *Insider threat tools*, 2017.

[25] John Rupert Firth, *Applications of general linguistics*, Transactions of the Philological Society **56** (1957), no. 1, 1–14.

[26] Gaurang Gavai, Kumar Sricharan, Dave Gunning, John Hanley, Mudita Singhal, and Rob Rolleston, *Supervised and unsupervised methods to detect insider threat from enterprise social and online activity data.*, JoWUA **6** (2015), no. 4, 47–63.

[27] Gaurang Gavai, Kumar Sricharan, Dave Gunning, Rob Rolleston, John Hanley, and Mudita Singhal, *Detecting insider threat from enterprise social and online activity data*, Proceedings of the 7th ACM CCS international workshop on managing insider security threats, ACM, 2015, pp. 13–20.

[28] Gemalto, *Breach level index data breach database and risk assessment calculator*, 2016.

[29] Joshua Glasser and Brian Lindauer, *Bridging the gap: A pragmatic approach to generating insider threat data*, 2013 IEEE Security and Privacy Workshops, IEEE, 2013, pp. 98–104.

[30] Yoav Goldberg, *A primer on neural network models for natural language processing*, Journal of Artificial Intelligence Research **57** (2016), 345–420.

[31] Markus Goldstein and Seiichi Uchida, *A comparative evaluation of unsupervised anomaly detection algorithms for multivariate data*, PloS one **11** (2016), no. 4, e0152173.

[32] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio, *Deep learning*, vol. 1, MIT press Cambridge, 2016.

[33] Palash Goyal, Sumit Pandey, and Karan Jain, *Deep learning for natural language processing*, vol. 27, 2018.

[34] Nizar Grira, Michel Crucianu, and Nozha Boujemaa, *Unsupervised and semi-supervised clustering: a brief survey*, A review of machine learning techniques for processing multimedia content **1** (2004), 9–16.

[35] Guo Haixiang, Li Yijing, Jennifer Shang, Gu Mingyun, Huang Yuanyue, and Gong Bing, *Learning from class-imbalanced data: Review of methods and applications*, Expert Systems with Applications **73** (2017), 220–239.

[36] Zellig S Harris, *Distributional structure*, Word **10** (1954), no. 2-3, 146–162.

[37] Satu Helske and Jouni Helske, *Mixture hidden markov models for sequence data: the seqhmm package in r*, arXiv preprint arXiv:1704.00543 (2017).

[38] Sepp Hochreiter and Jürgen Schmidhuber, *Long short-term memory*, Neural computation **9** (1997), no. 8, 1735–1780.

[39] Ivan Homoliak, Flavio Toffalini, Juan Guarnizo, Yuval Elovici, and Martín Ochoa, *Insight into insiders and it: A survey of insider threat taxonomies, analysis, modeling, and countermeasures*, ACM Computing Surveys (CSUR) **52** (2019), no. 2, 30.

[40] The White House, *Presidential Memorandum – National Insider Threat Policy and Minimum Standards for Executive Branch Insider Threat Programs*, 2012.

[41] Carly L. Huth, David W. Chadwick, William R. Claycomb, and Ilsun You, *Guest editorial: A brief overview of data leakage and insider threats*, Information Systems Frontiers **15** (2013), no. 1, 1–4.

[42] Sergey Ioffe and Christian Szegedy, *Batch normalization: Accelerating deep network training by reducing internal covariate shift*, arXiv preprint arXiv:1502.03167 (2015).

[43] Ruchi Jain and Nasser S Abouzakhar, *Hidden markov model based anomaly intrusion detection*, Internet Technology And Secured Transactions, 2012 International Conference for, IEEE, 2012, pp. 528–533.

[44] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani, *An introduction to statistical learning*, vol. 112, Springer, 2013.

[45] S Jha, L Kruger, T Kurtx, Y Lee, and A Smith, *A filtering approach to anomaly and masquerade detection*, University of Wisconsin, Tech. Rep (2004).

[46] Timothy E. Heron John O. Cooper and William L. Heward, *Applied behavior analysis*, Pearson Education Inc., 2014.

[47] Wen-Hua Ju and Yehuda Vardi, *A hybrid high-order markov chain model for computer intrusion detection*, Journal of Computational and Graphical Statistics **10** (2001), no. 2, 277–295.

[48] Parisa Kaghazgaran and Hassan Takabi, *Toward an insider threat detection framework using honey permissions.*, J. Internet Serv. Inf. Secur. **5** (2015), no. 3, 19–36.

[49] Bartosz Krawczyk, *Learning from imbalanced data: open challenges and future directions*, Progress in Artificial Intelligence **5** (2016), no. 4, 221–232.

[50] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton, *Deep learning*, nature **521** (2015), no. 7553, 436.

[51] Yann A LeCun, Léon Bottou, Genevieve B Orr, and Klaus-Robert Müller, *Efficient backprop*, Neural networks: Tricks of the trade, Springer, 2012, pp. 9–48.

[52] Do-hyeon Lee, Doo-young Kim, and Jae-il Jung, *Multi-stage intrusion detection system using hidden markov model algorithm*, Information Science and Security, 2008. ICISS. International Conference on, IEEE, 2008, pp. 72–77.

[53] L Li and Constantine N Manikopoulos, *Windows nt one-class masquerade detection*, Proceedings from the Fifth Annual IEEE SMC Information Assurance Workshop, 2004., IEEE, 2004, pp. 82–87.

[54] Yihua Liao and V Rao Vemuri, *Use of k-nearest neighbor classifier for intrusion detection*, Computers & security **21** (2002), no. 5, 439–448.

[55] Liu Liu, Olivier De Vel, Qing-Long Han, Jun Zhang, and Yang Xiang, *Detecting and preventing cyber insider threats: A survey*, IEEE Communications Surveys & Tutorials **20** (2018), no. 2, 1397–1417.

[56] Teresa F Lunt, *A survey of intrusion detection techniques*, Computers & Security **12** (1993), no. 4, 405–418.

[57] Marcus A Maloof and Gregory D Stephens, *Elicit: A system for detecting insiders who violate need-to-know*, International Workshop on Recent Advances in Intrusion Detection, Springer, 2007, pp. 146–166.

[58] Christopher Manning, Prabhakar Raghavan, and Hinrich Schütze, *Introduction to information retrieval*, Natural Language Engineering **16** (2010), no. 1, 100–103.

[59] Walaa Medhat, Ahmed Hassan, and Hoda Korashy, *Sentiment analysis algorithms and applications: A survey*, Ain Shams engineering journal **5** (2014), no. 4, 1093–1113.

[60] Kevin P Murphy, *Machine learning: a probabilistic perspective*, MIT press, 2012.

[61] Justin Myers, Michael R Grimaila, and Robert F Mills, *Towards insider threat detection using web server logs*, Proceedings of the 5th Annual Workshop on Cyber Security and Information Intelligence Research: Cyber Security and Information Intelligence Challenges and Strategies, ACM, 2009, p. 54.

[62] Chikashi Nobata, Joel Tetreault, Achint Thomas, Yashar Mehdad, and Yi Chang, *Abusive language detection in online user content*, Proceedings of the 25th international conference on world wide web, International World Wide Web Conferences Steering Committee, 2016, pp. 145–153.

[63] Christopher Olah, *Understanding lstm networks*, (2015).

[64] Won Park, Youngin You, and Kyungho Lee, *Detecting potential insider threat: Analyzing insiders' sentiment exposed in social media*, Security and Communication Networks **2018** (2018).

[65] Younghee Park and Salvatore J Stolfo, *Software decoys for insider threat*, Proceedings of the 7th ACM Symposium on Information, Computer and Communications Security, ACM, 2012, pp. 93–94.

[66] Pallabi Parveen and Bhavani Thuraisingham, *Unsupervised incremental sequence learning for insider threat detection*, 2012 IEEE International Conference on Intelligence and Security Informatics, IEEE, 2012, pp. 141–143.

[67] Animesh Patcha and Jung-Min Park, *An overview of anomaly detection techniques: Existing solutions and latest technological trends*, Computer networks **51** (2007), no. 12, 3448–3470.

[68] LLC Ponemon Institue, *Cost of cyber crime study: United states*, Traverse City (2013).

[69] Sasanka Potluri and Christian Diedrich, *Accelerated deep neural networks for enhanced intrusion detection system*, 2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA), IEEE, 2016, pp. 1–8.

[70] S Pramanick, S Rajagopalan, and Eric van den Berg, *Mitigating the insider threat with high-dimensional anomaly detection*, Tech. report, TELCORDIA TECHNOLOGIES INC MORRISTOWN NJ, 2004.

[71] Christian W Probst, Jeffrey Hunker, Matt Bishop, and Dieter Gollmann, *Insider threats in cyber security*, vol. 49, Springer, 2010.

[72] Lawrence R Rabiner, *A tutorial on hidden markov models and selected applications in speech recognition*, Readings in speech recognition, Elsevier, 1990, pp. 267–296.

[73] Tabish Rashid, Ioannis Agrafiotis, and Jason RC Nurse, *A new take on detecting insider threats: exploring the use of hidden markov models*, Proceedings of the 8th ACM CCS International Workshop on Managing Insider Security Threats, ACM, 2016, pp. 47–56.

[74] Robert Richardson and CSI Director, *Csi computer crime and security survey*, Computer security institute **1** (2008), 1–30.

[75] James Rosindell and Yan Wong, *Biodiversity, the tree of life, and science communication*, Phylogenetic Diversity, Springer, 2018, pp. 41–71.

[76] M Ben Salem and Salvatore J Stolfo, *Masquerade attack detection using a search-behavior modeling approach*, Columbia University, Computer Science Department, Technical Report CUCS-027-09 (2009).

[77] Malek Ben Salem, Shlomo Hershkop, and Salvatore J Stolfo, *A survey of insider attack detection research*, Insider Attack and Cyber Security, Springer, 2008, pp. 69–90.

[78] Malek Ben Salem and Salvatore J Stolfo, *Decoy document deployment for effective masquerade attack detection*, International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment, Springer, 2011, pp. 35–54.

[79] Anna Schmidt and Michael Wiegand, *A survey on hate speech detection using natural language processing*, Proceedings of the Fifth International Workshop on Natural Language Processing for Social Media, 2017, pp. 1–10.

[80] E Eugene Schultz, *A framework for understanding and predicting insider attacks*, Computers & Security **21** (2002), no. 6, 526–531.

[81] R Socher and C Manning, *Deep learning for natural language processing (without magic)*, Keynote at the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL 2013), 2013.

[82] Stephen V Stehman, *Selecting and interpreting measures of thematic classification accuracy*, Remote sensing of Environment **62** (1997), no. 1, 77–89.

[83] Svetlana Symonenko, Elizabeth D Liddy, Ozgur Yilmazel, Robert Del Zoppo, Eric Brown, and Matt Downey, *Semantic analysis for monitoring insider threats*, International Conference on Intelligence and Security Informatics, Springer, 2004, pp. 492–500.

[84] Boleslaw K Szymanski and Yongqiang Zhang, *Recursive data mining for masquerade detection and author identification*, Proceedings from the Fifth Annual IEEE SMC Information Assurance Workshop, 2004., IEEE, 2004, pp. 424–431.

[85] Hassan Takabi and J Haadi Jafarian, *Insider threat mitigation using moving target defense and deception*, Proceedings of the 2017 International Workshop on Managing Insider Security Threats, ACM, 2017, pp. 93–96.

[86] Paul J Taylor, Coral J Dando, Thomas C Ormerod, Linden J Ball, Marisa C Jenkins, Alexandra Sandham, and Tarek Menacere, *Detecting insider threats through language change.*, Law and human behavior **37** (2013), no. 4, 267.

[87] E Ted, Henry G Goldberg, Alex Memory, William T Young, Brad Rees, Robert Pierce, Daniel Huang, Matthew Reardon, David A Bader, Edmond Chow, et al., *Detecting insider threats in a real corporate database of computer usage activity*, Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining, ACM, 2013, pp. 1393–1401.

[88] Theano, *Gradient kernel description*, `http://deeplearning.net/software/theano/extending/op.html?highlight=grad#grad`, 2018, Accessed: 2018-02-12.

[89] Paul Thompson, *Weak models for insider threat detection*, Sensors, and Command, Control, Communications, and Intelligence (C3I) Technologies for Homeland Security and Homeland Defense III, vol. 5403, International Society for Optics and Photonics, 2004, pp. 40–49.

[90] Aaron Tuor, Samuel Kaplan, Brian Hutchinson, Nicole Nichols, and Sean Robinson, *Deep learning for unsupervised insider threat detection in structured cybersecurity data streams*, arXiv preprint arXiv:1710.00811 (2017).

[91] Nikos Virvilis, Bart Vanautgaerden, and Oscar Serrano Serrano, *Changing the game: The art of deceiving sophisticated attackers*, 2014 6th International Conference On Cyber Conflict (CyCon 2014), IEEE, 2014, pp. 87–97.

[92] Florian Walch, *Deep Learning for Image-Based Localization*, arXiv preprint (2016).

[93] Wei Wang, Jeffrey Bickford, Ilona Murynets, Ramesh Subbaraman, Andrea G Forte, and Gokul Singaraju, *Catching the wily hacker: A multilayer deception system*, 2012 35th IEEE Sarnoff Symposium, IEEE, 2012, pp. 1–6.

[94] Wikipedia, *Neuron — Wikipedia, the free encyclopedia*, 2010, [Online; accessed 10-September-2019].

[95] Robert Willison and Merrill Warkentin, *Motivations for employee computer crime: understanding and addressing workplace disgruntlement through the application of organisational justice*, Proceedings of the IFIP TC8 International Workshop on Information Systems Security Research. International Federation for Information Processing, 2009, pp. 127–144.

[96] Ting-Fang Yen, Alina Oprea, Kaan Onarlioglu, Todd Leetham, William Robertson, Ari Juels, and Engin Kirda, *Beehive: Large-scale log analysis for detecting suspicious activity in enterprise networks*, Proceedings of the 29th Annual Computer Security Applications Conference, ACM, 2013, pp. 199–208.

[97] Ozgur Yilmazel, Svetlana Symonenko, and Niranjan Balasubramanian, *Leveraging one-class svm and semantic analysis to detect anomalous content*, International Conference on Intelligence and Security Informatics, Springer, 2005, pp. 381–388.

[98] Ozgur Yilmazel, Svetlana Symonenko, Niranjan Balasubramanian, and Elizabeth D Liddy, *Improved document representation for classification tasks for the intelligence community.*, AAAI Spring Symposium: AI Technologies for Homeland Security, 2005, pp. 76–82.

[99] William T Young, Alex Memory, Henry G Goldberg, and Ted E Senator, *Detecting unknown insider threat scenarios*, Security and Privacy Workshops (SPW), 2014 IEEE, IEEE, 2014, pp. 277–288.

[100] Fangfang Yuan, Yanan Cao, Yanmin Shang, Yanbing Liu, Jianlong Tan, and Binxing Fang, *Insider threat detection with deep neural network*, International Conference on Computational Science, Springer, 2018, pp. 43–54.

[101] Lei Zhang, Shuai Wang, and Bing Liu, *Deep learning for sentiment analysis: A survey*, Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery **8** (2018), no. 4, e1253.

[102] Ye Zhang and Byron Wallace, *A sensitivity analysis of (and practitioners' guide to) convolutional neural networks for sentence classification*, arXiv preprint arXiv:1510.03820 (2015).

[103] Arthur Zimek, Ricardo JGB Campello, and Jörg Sander, *Ensembles for unsupervised outlier detection: challenges and research questions a position paper*, Acm Sigkdd Explorations Newsletter **15** (2014), no. 1, 11–22.